VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
HORNICKO-GEOLOGICKÁ FAKULTA
Katedra geoinformatiky

Paralelní výpočty pro analýzu viditelnosti

DIPLOMOVÁ PRÁCE

Autor: **Bc. Dominika Foldynová**
Vedoucí diplomové práce: **Ing. Kateřina Růžičková, Ph.D.**

Ostrava 2019

VŠB - Technická univerzita Ostrava
Hornicko-geologická fakulta
Katedra geoinformatiky

Zadání diplomové práce

Student:	**Bc. Dominika Foldynová**
Studijní program:	N3654 Geodézie, kartografie a geoinformatika
Studijní obor:	3608T002 Geoinformatika
Téma:	Paralelní výpočty pro analýzu viditelnosti Parallel Processing for Visibility Analysis
Jazyk vypracování:	čeština

Zásady pro vypracování:

- Studium problematiky náročnosti výpočtů při analýzách viditelnosti (identifikace, které výpočtové úlohy jsou časově náročné a proč)
- Studium možností programové paralelizace výpočtů
- Výběr metody paralelizace pro vybrané úlohy analýzy viditelnosti
- Implementace vybrané metody paralelizace pro vybrané úlohy analýzy viditelnosti
- Ověření využití vybrané/implementované metody paralelizace na testovacích datech
- Zhodnocení dosažených výsledků

Rozsah grafických prací:
dle potřeby

Rozsah původní zprávy:
50 - 70 normostran textu

Formální náležitosti diplomové práce stanoví směrnice děkana HGF HGF_SME_15_001 Pokyny pro zpracování závěrečných prací, zveřejněné na webových stránkách fakulty - https://www.hgf.vsb.cz/csold/portal-iso/interni/platne-dokumenty/sme/HGF_SME_15_001_A.pdf.

Seznam doporučené odborné literatury:

Rášová, A. Pravdepodobná viditeľnosť a fuzzy viditeľnosť: modelovanie neistoty a neurčitosti analýz viditeľností. Diplomová práce. STU v Bratislave, Stavebná fakulta. 2013.
Růžičková, K., Dohnalová, M., Růžička, J.: Sensitivity analysis of analysis of visibility form line. GIS Ostrava 2012: Surface models for geosciences. 23-25 January 2012, Ostrava. pp 225-239. ISBN 978-80248-2667-7, ISSN 1213-2454.
Teng, Y. A., Dementhon, D., Davis, L.S. Region-to-region visibility analysis using data parallel machines, Concurrency: Practice and Experience, vol. 5, pp 379–406, 1993.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Kateřina Růžičková, Ph.D.**

Datum zadání: 31.10.2018

Datum odevzdání: 30.04.2019

doc. Ing. Michal Kačmařík, Ph.D.
vedoucí katedry

prof. Ing. Vladimír Slivka, CSc., dr.h.c.
děkan fakulty

Prohlášení

- *Celou diplomovou práci včetně příloh jsem vypracovala samostatně a uvedla jsem všechny použité podklady a literaturu.*
- *Byla jsem seznámena s tím, že na moji diplomovou práci se plně vztahuje zákon č.121/2000 Sb. - autorský zákon*
- *Beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3).*
- *Souhlasím s tím, že jeden výtisk diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o diplomové práci, obsažené v anotaci, budou zveřejněny v informačním systému VŠB-TUO.*
- *Rovněž souhlasím s tím, že kompletní text diplomové práce bude publikován v materiálech zajišťujících propagaci VŠB-TUO, vč. příloh časopisů, sborníků z konferencí, seminářů apod. Publikování textu práce bude provedeno v omezeném rozlišení, které bude vhodné pouze pro čtení a neumožní tedy případnou transformaci textu a dalších součástí práce do podoby potřebné pro jejich další elektronické zpracování.*
- *Bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona.*
- *Bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).*

V Ostravě dne 29. 4. 2019 ***Dominika Foldynová***

Poděkování

Především bych ráda poděkovala mé vedoucí diplomové práce Ing. Kateřině Růžičkové, Ph.D. za veškerou její pomoc, rady a vedení, a také mému kamarádovi Bc. Jiřímu Krasulovi za podporu a pomoc s programováním.

Nicméně velké díky patří mým rodičům za veškerou podporu, pomoc a zázemí.

ANOTACE BAKALÁŘSKÉ PRÁCE

Tato práce se zabývá možnostmi paralelizace náročných výpočtů analýz viditelnosti. Konkrétně byly řešeny úlohy viditelnosti z linie a stochastické vyhodnocení viditelnosti. Úlohy byly paralelizovány za účelem urychlení celkového výpočtu. Dosažené výsledky byly srovnány a výsledky paralelních výpočtů vyhodnoceny na základě hodnotících parametrů pro paralelní algoritmy.

Klíčová slova: paralelní programování, procesor, vlákno, analýza viditelnosti.

ANOTATION OF THESIS

The thesis deals with parallelization options of the difficult calculations of visibility analysis. Specifically, there were solved two tasks – the visibility from the line and the stochastic visibility evaluation. The tasks were parallelized to accelerate the overall calculation. The results were compared and the results of parallel calculations were evaluated based on evaluation parameters for parallel algorithms.

Keywords: parallel programming, processor, thread, visibility analysis.

OBSAH

SEZNAM ZKRATEK A POJMŮ

3D	Trojrozměrný
DMT	Digitální model terénu
DMP	Digitální model povrchu
GIS	Geoinformační systém
GRASS	Geographic Resources Analysis Support System

1 ÚVOD

Informační technologie zaznamenaly v posledních letech neskutečný posun. Nároky na výpočetní výkon počítačových technologií s postupem času rostl, a proto bylo nezbytné nalézt řešení pro urychlení výpočetních operací. Jedno z řešení, jak urychlit výpočet operací je právě tzv. paralelní programování, které přichází s návody, postupy a algoritmy, jak výpočetní operaci urychlit. Paralelní programování vychází z předpokladu, že ne všechny operace zpracovávané výpočetní jednotkou jsou na sobě závislé, a proto nemusí být vykonávány sériově za sebou. Diplomová práce se zabývá oběma způsoby programování (sériové i paralelní), jejímž cílem je porovnání rychlosti výpočtů sériového programování s rychlostí výpočtů programování paralelního na základě sledovaných parametrů u počítačů, na kterých byly výpočty prováděny.

V druhé kapitole jsou popsány základy paralelního programování. Úvodní část kapitoly se zabývá Amdahlovým zákonem, který popisuje, jakého zrychlení lze dosáhnout paralelním zpracováním úlohy a Gustafsonovým zákonem, který popisuje, jakého zrychlení lze dosáhnout u výpočtů obsahujících velké datové bloky. Uvedené zákony jsou matematicky odvozeny a popsány. Následně je kapitola věnována technickému řešení paralelního programování, existujícím možnostem paralelismu ve více vláknech nebo na více procesorech, typům pamětí využívající počítače při paralelním programováním a příklady paralelního programování z praxe. Třetí kapitola je věnována jednotlivým metodám analýz viditelnosti a náročnosti jejich výpočtu v závislosti na typu zpracovávané úlohy.

Praktická studie se zabývá praktickými úlohami, které byly naprogramovány oběma zmíněnými způsoby (sériově i paralelně). První řešenou úlohou je viditelnost z linie vycházející ze závěrečné práce, jejímž autorem je V. Lelek (2016). A druhá úloha se zabývá stochastickým vyhodnocením viditelnosti metodou Monte Carlo, která vychází z řešené úlohy v rámci cvičení předmětu Modelování a simulace v geovědách. Pro obě úlohy byl navržen vhodný způsob paralelizace a následně aplikován. Obě výpočetní úlohy byly vypočítány na školních počítačích s odlišnou konfigurací. Získané výsledky byly porovnány mezi sebou a vyhodnoceny na základě vybraných parametrů hodnocení.

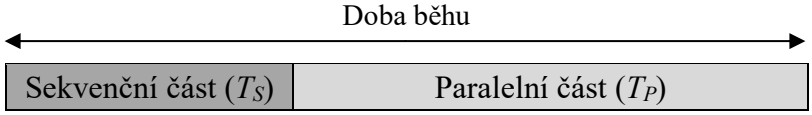
2 PARALELNÍ VÝPOČTY

Standartní program napsaný sekvenčním způsobem se také sekvenčně vykonává. To znamená, že zdrojový kód programu se vykonává řádek po řádku a nezačne se vykonávat další blok, dokud se nedokončí blok předchozí. Některé části programu však lze provést současně, nezávisle na sobě, a přitom efektivně využít výkon celého vícejádrového procesoru. Na tomto principu jsou založené tzv. paralelní výpočty. Cílem paralelního počítání je výrazné zkrácení času běhu programu.

Program by měl být napsán tak, aby se na vícejádrových procesorech zrychloval úměrně počtu jader. Ale stejně tak může být napsána vícevláknová aplikace na procesoru s jedním jádrem, protože i tento způsob je paralelní počítání s tím rozdílem, že zrychlení programu se neprojeví natolik, co na procesorech s více jádry. (Laryš, 2011)

Paralelní programování je tedy možnost zpracování několika operací (činností) souběžně. Ne vždy ale platí, že čím větší bude paralelizace, tím rychleji bude program zpracován. To znamená, že zde neplatí přímá úměra mezi počtem výpočetních jednotek a rychlostí zpracování. Urychlení výpočtu (angl. Speedup) – *S* je obvykle vyhodnocováno jako poměr doby výpočtu nejlepšího sekvenčního programu a doby výpočtu paralelního programu *P* na témže počítači.

Kvalitní paralelní program musí nepochybně vykazovat náležité urychlení při dostatečném využití procesorů. Dosažitelné urychlení je pro určitý program omezeno tzv. *Amdahlovým zákonem*. Amdahlův zákon je nejjednodušší formou škálovacího zákona a usuzuje, že určitá část programu *S* musí být provedena sekvenčně, protože výpočet nelze zcela paralelizovat viz obrázek 1. (Gove, 2011)



Obr. 1 Doba běhu aplikace s jedním vláknem (Gove, 2011)

$$\text{Doba běhu} = T_S + T_P$$

Doba běhu paralelní aplikace, která využívá *N* procesorů, by vypadala následovně:

$$\text{Doba běhu} = T_S + \frac{T_P}{N}$$

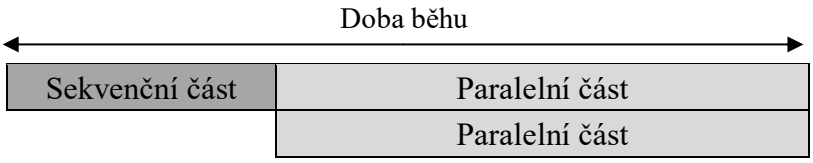
Pro výpočet, který probíhá na *N* procesorech je dosažitelné urychlení *S* omezeno dle následujícího vzorce –

$$S = \frac{1}{f+\frac{1-f}{N}} < \frac{1}{f}$$

Kde *f* je poměr času stráveného výpočtem sekvenční části ku celkovému času (na paralelním stroji) –

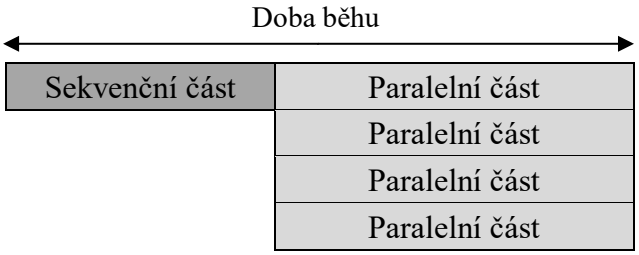
$$f = \frac{T_S}{T_S + T_P}$$

Jestliže paralelní část kódu využívá dvě vlákna, pak se doba běhu této paralelizované části výpočtu zkrátí na polovinu a výsledná aktivita procesoru by vypadala dle obrázku 2.



Obr. 2 Doba běhu aplikace se dvěma vlákny (Gove, 2011)

Doba běhu při využívání čtyř vláken by vypadala následovně:



Obr. 3 Doba běhu aplikace se čtyřmi vlákny (Gove, 2011)

Se vzrůstajícím počtem procesorů začíná výkon určovat sekvenční část aplikace, protože platí, že program nemůže běžet rychleji, než jak dlouho trvá sekvenční část. To znamená, že v určitém bodě programu přidání dalších vláken nezpůsobí znatelný rozdíl v celkové době běhu aplikace.

Maximální efektivní počet vláken lze stanovit v souvislosti se zmíněným Amdahlovým zákonem – doba běhu $= T_C + \frac{T_P}{N}$, kde T_C je optimální doba běhu při nekonečném počtu vláken. Doba běhu v rámci T procentního optima je (Gove, 2011):

$$\text{Přijatelná doba běhu} = T_C * (1 + T)$$

Následně lze porovnat přijatelnou dobu běhu s dobou běhu s *N* vlákny:

$$T_C * (1 + T) = (T_C + \frac{T_P}{N})$$

Jestliže je paralelizace prováděna na složitějších architekturách je nutné detailněji sledovat celkový čas výpočtu. Celkový čas výpočtu sekvenčního algoritmu je doba, která uplyne od spuštění výpočtu do jeho ukončení. A celkový čas výpočtu paralelního algoritmu je doba, která uplyne od spuštění výpočtu do doby, kdy skončí poslední paralelní výpočet. Paralelní doba výpočtu zahrnuje distribuci vstupních i sběr výstupních dat. U celkového času výpočtu algoritmu lze obvykle rozlišovat následující hlavní položky:

- **výpočetní čas** – je to doba, po kterou je prováděn vlastní výpočet. Při určitém zjednodušení se jedná o dobu běhu sekvenčního programu určující výpočetní čas. Zejména závisí na použitém procesoru a zahrnuje aritmetické, logické a paměťové operace.
- **komunikační čas** – je doba strávená komunikací, která je potřebná pro vysílání a příjem zpráv. Závisí především na velikosti zprávy a na vlastnostech komunikačního subsystému. Komunikace mezi vlákny nebo mezi procesy je nezbytná. Příkladem je situace, kdy jedno vlákno může signalizovat ostatním čekajícím vláknům, že inicializace je hotová. Nicméně vlákna nevyžadují žádný speciální komunikační mechanismus z důvodu společného paměťového prostoru.
- **jalový čas** – je doba, po kterou proces čeká na nějaký impuls, aby mohl dále pokračovat ve výpočtu. Proces pokračuje, jakmile doběhne předchozí část programu. Jalový čas se těžce kvantifikuje, a proto často bývá ve výkonnostních modelech zanedbáván. (Jakl, 2011)

Čas, který je potřebný pro zakládání nebo rušení procesů je většinou jednorázový a projevuje se především u méně objemných výpočtů. Proto většinou není zahrnován do celkového času běhu paralelního výpočtu. Matějovic et al. (1997, s. 32)

Rozšířením Amdahlova zákona je *Gustavsonův zákon*, který je založen na efektivním paralelním zpracováním výpočtů obsahujících velké datové bloky. Gustavsonův zákon odpovídá na nedostatky Amdahlova zákona, který nedokáže zcela popsat situaci při zapojení více strojů. Gustavsonův zákon je definován následovně:

$$S = M - \alpha(M - 1),$$

kde S je zrychlení, M je počet mikroprocesorů a α je největší část, kterou nelze paralelizovat. Platí, že čím větší je α, tím je maximální teoretické zrychlení S menší. (Polášek, 2016)

2.1 Parametry hodnocení paralelních algoritmů

Jeden z mnoha hodnotících parametrů paralelních algoritmů, který již byl zmíněn výše je paralelní čas T_p, který představuje čas, jež uplyne od začátku paralelního výpočtu do ukončení výpočtu na posledním procesoru. Dalšími hodnotícími parametry jsou např.:

- *Paralelní cena* – je parametr odpovídající součinu paralelního času *T(n,p)* a počtu procesorů *p*.

$$C(n,p) = p \times T(n,p)$$

- *Paralelní efektivnost E(n,p)* – je poměrem sekvenční a paralelní složitosti (ceny) –

$$E(n,p) = \frac{SU(n)}{C(n,p)}$$

  Jinak řečeno paralelní efektivnost je zrychlení na 1 procesor. Závislost efektivity na rostoucím počtu procesorů *p* plyne z Amdahlova zákona.

- *Paralelní práce – W(n,p)* odpovídá počtu paralelně provedených operací na *p* procesorech.

$$W(n,p) = p_1 + p_2 +_{\dots} p_i$$

  Kde p_i je počet aktivně pracujících procesorů.

- *Paralelní režie – H(n,p)* je parametr, který určuje rozdíl mezi paralelní cenou a časem nejrychlejšího známého sekvenčního algoritmu *SU(n)*.

$$H(n,p) = C(n,p) - SU(n)$$

Mezi hodnotící parametry lze zařadit i tzv. paralelní zpomalení, protože ne vždy zavedená paralelizace je efektivnější než sekvenční řešení. Paralelní výpočet může být pomalejší než klasický sekvenční výpočet z důvodu vzájemné komunikace mezi vlákny či procesy do kterých je úloha rozložena. Objem vzájemné komunikace může růst exponenciálně vůči počtu vláken při nevhodně zvoleném postupu výpočtu. Z tohoto důvodu může docházet ke zpomalení výpočtu.

2.2 Technické řešení

V případě paralelního programování lze rozlišovat dva případy paralelismu (Faigl, 2016):

1. **Hardwarový paralelismus** – jedná se o multiprocesorový systém, kde každý proces běží na svém procesoru,
2. **Softwarový paralelismus** – je tzv. pseudoparalelismus, kde procesy běží paralelně na jednom procesorovém systému (*multithreding viz kap. 2.3*).

Pro vytváření paralelní aplikace je nezbytná znalost fungování hardwaru, i když využití více jader je spíše problém softwarový než hardwarový. V případě hardwarového paralelismu se jedná o víceprocesorovou aplikaci, zatímco v případě softwarového paralelismu se jedná o vícevláknovou aplikaci.

V rámci této diplomové práce byl při vytváření paralelních výpočtů aplikován softwarový paralelismus, neboť jeho výhodou je schopnost jednoho vlákna ukládat takový prvek dat do paměti, který se stává okamžitě viditelný pro všechna ostatní vlákna v daném procesu. Další výhodou softwarového paralelismu je schopnost všech vláken sdílet totožné záznamy v paměti i mezipaměti. Nevýhodou v tomto případě může být situace, kdy selhání jednoho vlákna způsobí selhání celé aplikace, která v případě hardwarového paralelismu nemůže nastat, neboť každý procesor je izolovaný. (Gove, 2011)

2.3 Možnosti rozložení výpočtů (do více vláken)

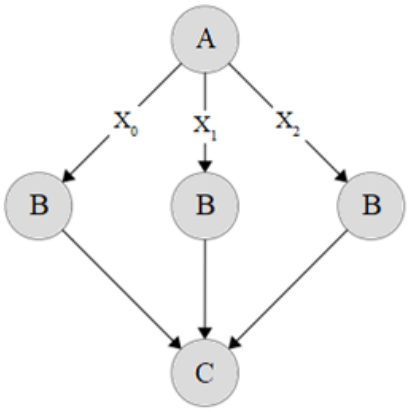
Původně procesor uměl zpracovávat pouze jedno vlákno. V dnešní době však lze vytvořit více vláken v jednom procesu. Tato metoda se nazývá *multithreading*. Při multithreadingu je procesor stále jeden, ale „předstírá“, že jich je více. Jednotlivá vlákna mohou vykonávat různé činnosti v rámci jednoho procesu. Vlákna jednoho procesu sdílí paměťový prostor a další systémové prostředky. Díky sdílené paměti dochází ke zjednodušení komunikace mezi vlákny, ale naopak zvyšuje riziko souběhu, které může nastat při chybném či nepředvídatelném pořadí nebo načasování jednotlivých operací při zpracování sdílených dat. Výsledky z paralelních vláken mohou být dále zpracovávány. Nezbytné je hlídat, zda zpracování ve všech vláknech doběhlo. Pokud ano, je možné následně spustit výpočet s výsledky z jednotlivých vláken.

Bohužel i současné nejlepší procesory dokáží celkem zpracovávat pouhých 32 vláken najednou, což je pro moderní aplikace využívající paralelní zpracování nedostatečné. Procesor navíc nezpracovává pouze výpočetní program, ale také musí udržovat procesy nutné pro běh systému. Z tohoto důvodu nelze zaměřit celou výpočetní kapacitu na zpracovávaný program. (Polášek, 2016)

Paralelismus ve více vláknech lze dělit do dvou kategorií označované jako *datový paralelismus* a *úlohový paralelismus*. Existuje řada přístupů a způsobů, jak rozdělit práci mezi více vláken. Mezi nejpoužívanější přístupy patří: (Gove, 2011)

a. **Datový paralelismus** – je založen na rozdělení velkého objemu dat na menší části, jejich následném paralelním zpracování a konečném složení výsledku. Výpočet je tzv. deterministický, to znamená, že výsledek bude vždy stejný. Výhodou datového paralelismu je, že neřeší rozdělení výpočtové úlohy na více částí, a tedy nemusí hlídat časovou návaznost jednotlivých výpočetních částí.

   Při hledání vhodného způsobu paralelizace je důležité vytvořit tzv. graf datové závislosti. Jedná se o orientovaný graf, který se skládá z uzlů a hran. Uzly představují úlohy prováděné nad datovými instancemi a hrany představují závislosti mezi úlohami. V případě datového paralelismu graf datové závislosti obsahuje nezávislé úlohy provádějící totožné operace nad různými instancemi dat. (Blizňák, 2014)



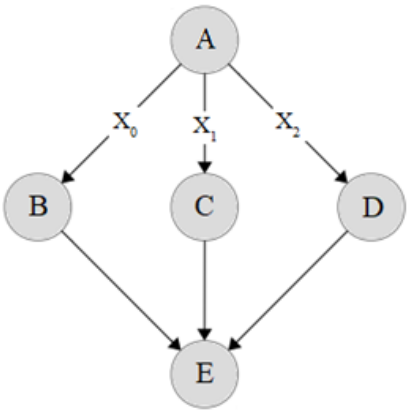
Obr. 4 Datový paralelismus (Blizňák, 2014)

Kde A je příprava/rozdělení dat na jednotlivé části, B představuje řešení výpočetní úlohy a C odpovídá spojení výsledků řešené úlohy ze všech výpočetních částí.

Příkladem datového paralelismu může být agregace výskytu např. listnatých a jehličnatých lesů v jednotlivých územně správních jednotkách souběžně pro jednotlivé lokality samostatných sad dat.

Tento vzor lze rovněž označit za fork-join, kde fork je rozdělení práce mezi jednotlivá vlákna a join je bod, v němž se všechna vlákna synchronizují po dokončení přiřazené práce. (Gove, 2011)

b. **Funkcionální paralelismus** (*Úlohový paralelismus*) – je založen na souběžném zpracování více řešených úloh různými operacemi. Graf závislosti dat se skládá z nezávislých úloh provádějící různé operace nad různými instancemi dat.



Obr. 5 Funkcionální paralelismus (Blizňák, 2014)

Kde A je příprava vstupních dat, B, C a D jsou nezávislé výpočetní úlohy a E je souhrnné vyhodnocení úlohy.

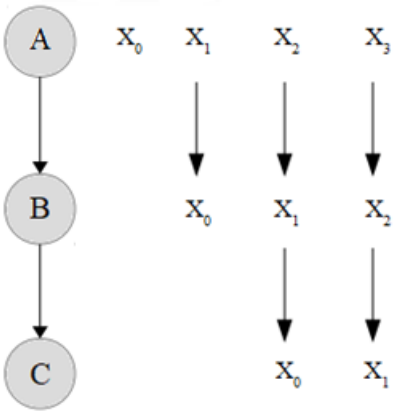
Příkladem funkcionálního paralelismu z běžného života může být situace, kdy rodina přijede na víkendový pobyt na chatu a po příjezdu se každý člen rodiny zhostí jednoho z úkolů – otec poseká dříví, matka zamete dvůr a děti si hrají na zahradě.

Příkladem funkcionálního paralelismu v rámci geoinformatiky, může být multikriteriální analýza zahrnující 3 kritéria (např. sklon DMR, vzdálenost od silnice a výběr typů půdy) pro rozhodování. Z toho vyplývá, že každé vlákno má jinou funkcionalitu (na Obr. 3 – B, C, D) a pracuje s jinými daty. V E se poté vyhodnotí míra splnění všech 3 kritérií souhrnně.

c. **Pipelining** (*Instrukční fronty*) – se podobá výrobní lince a umožňuje zpracovávat instrukce rozdělené na více kroků, které jsou jednodušší na provedení a dále tyto instrukce řetězit. Pipelining se potýká se třemi následujícími typy závislostí, jejichž nevýhodou je snížení efektivity výpočtu:

   - *Datová závislost* – nadejde v případě, kdy jedna instrukce vyžaduje data, která ještě nebyla vypočtena.
   - *Závislost ve zdrojích* – nastává tehdy, kdy dvě instrukce chtějí např. přistupovat na stejné místo v paměti.
   - *Závislost na podmínce* – nastává v případě, kdy není znám výsledek výpočtu, který může ovlivnit, jakým způsobem bude kód následně proveden.

Graf datové závislosti se skládá z jednoduché cesty či větve provádějící různé závislé operace nad jednou instancí dat.

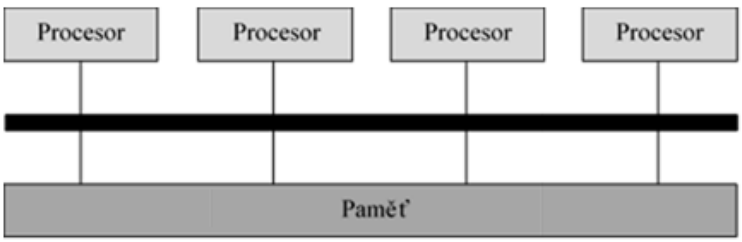


Obr. 6 Pipelining (Blizňák, 2014)

V řadě situací stačí jediná paralelizační strategie. Jestliže není jedna paralelizační strategie dostačující pro efektivní vyřešení problému, pak je nutné zvolit vhodnou kombinaci různých přístupů.

U počítačů využívajících některou ze zmíněných strategií paralelizace lze rozlišovat tři následující obecné typy pamětí (Polášek, 2016):

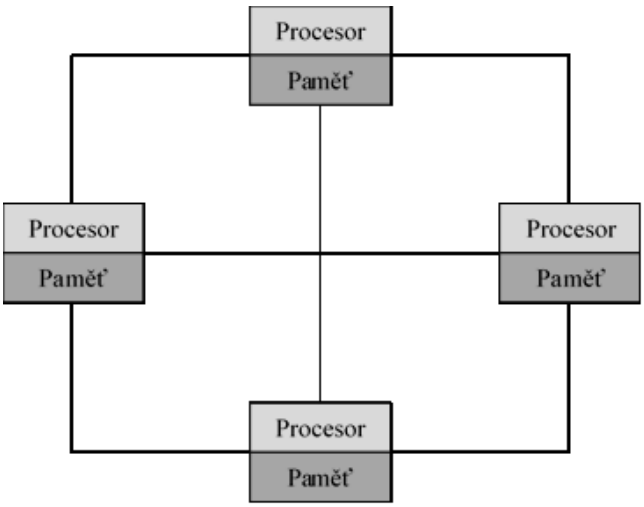
1. **Sdílená paměť** – všechny procesory, či jádra procesorů mají přístup k jediné společné paměti. Komunikace a synchronizace mezi jednotlivými úkoly běžícími na různých mikroprocesorech je prováděna zápisem a načtením ze společné sdílené paměti. Systémy využívající sdílenou paměť tvoří velkou část paralelního programování.



Obr. 7 Systém se sdílenou pamětí (Faigl, 2016)

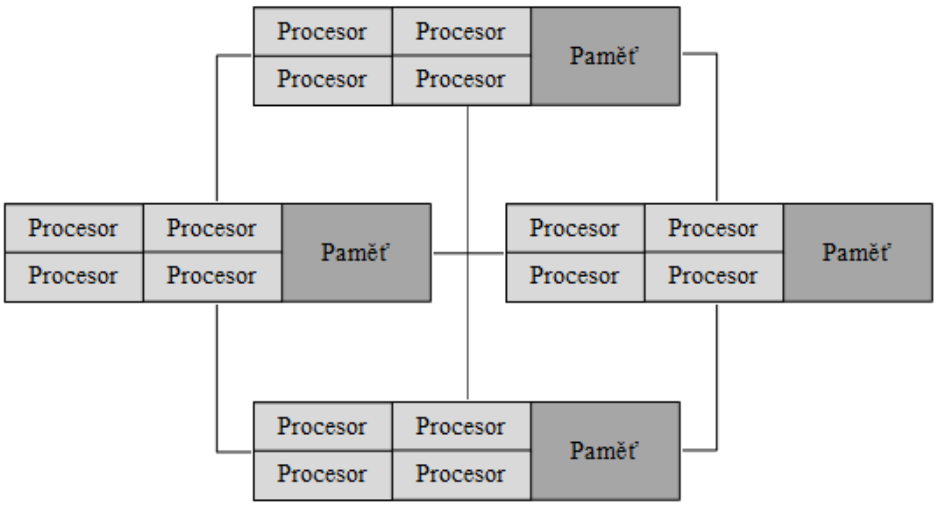
2. **Distribuovaná paměť** – jednotlivé mikroprocesory mají svůj paměťový prostor, na rozdíl od sdílené paměti, kde všechny mikroprocesory pracují se sdíleným paměťovým prostorem. Výpočetní úlohy mohou pracovat jen s lokální pamětí. Jestliže jsou potřebná data uložena mimo lokální paměť, pak musí mikroprocesor komunikovat se vzdáleným mikroprocesorem, který má

potřebná data uložena ve své lokální paměti, protože mikroprocesor nemůže přistoupit sám k uloženým datům na lokální paměti jiného mikroprocesoru.



Obr. 8 Systém s distribuovanou pamětí (Faigl, 2016)

3. **Distribuovaná sdílená paměť** – je kombinace dvou předcházejících zmíněných typů pamětí. Na rozdíl od sdílené paměti, která se vyznačuje jedinou fyzickou centrální pamětí, distribuovaná sdílená paměť vyjadřuje jeden sdílený adresový prostor. Stejná fyzická adresa na dvou rozdílných mikroprocesorech vyjadřuje stejné místo v paměti.



Obr. 9 Systém s distribuovanou sdílenou pamětí

2.4 Alternativní řešení

Alternativou vícevláknové paralelizace, kde výpočet je rozložen do více vláken, je výpočet rozložený do více procesorů – víceprocesorová paralelizace (*multiprocesorový systém viz kap. 2.2*). Multiprocesorový systém (angl. Multiprocessing) je počítačová architektura, která ve své struktuře využívá více než jeden procesor pod dohledem operačního systému (na rozdíl od vícevláknové paralelizace), který umožňuje současný běh několika procesů. Nevýhodou alternativního řešení víceprocesorové paralelizace by bylo možné využití výpočtů pouze pro operační systém GNU/Linux.

Multiprocesorové systémy lze dělit dle několika kritérií. Základní dělení multiprocesorových systémů vychází z prostorové symetrie na základě softwarového či hardwarového hlediska. Multiprocesorové systémy z hlediska prostorové symetrie dělíme na (Dumek, 2013):

- **Symetrický multiprocessing –** znamená spolupráci dvou nebo více naprosto shodných procesorů v rámci jednoho počítače. Tyto procesory jsou naprosto shodné a mají společnou operační paměť. Na každém z procesorů běží část operačního systému. V případě symetrického multiprocessingu může se systémovými strukturami pracovat více procesorů.
- **Asymetrický multiprocessing** – se liší od symetrického multiprocessingu v tom, že procesor nemusí být stejného typu. Řízení zajišťuje centrální procesor a pouze jeden procesor může pracovat se systémovými datovými strukturami. V tomto případě je pak výhodou, že není potřeba zajišťovat sdílení systémových struktur. Nevýhodou je však nižší výkonnost na rozdíl od symetrického multiprocessingu.
- **Buňkový multiprocessing** – spočívá v rozdělení procesorů do samostatných počítačových prostředí s různými operačními systémy sdílející společnou paměť. Jedná se o heterogenní serverovou technologii od společnosti Unisys, která může spustit libovolnou kombinaci operačních systémů.

2.5 Příklady paralelizace

Paralelizace výpočtů se využívá ze tří důvodů:

1. *Časové náročnosti* – která nastává tehdy, když sekvenční výpočet na jednom procesoru trvá příliš dlouho, např. z důvodu zpracovávání velkého množství dat (dat z velkého území, velmi podrobná data, či zpracování dat ve 3D),
2. *Paměťové náročnosti* – která nastává v momentě, kdy výpočet vyžaduje více paměti, než procesor může poskytnout. Problém může nastat rovněž při zpracování velkého množství dat. (Prišť, 2013)
3. *Výpočetní náročnosti* – která nastává v případě složitosti algoritmů např. z důvodu náročných počtů, např. simulace pomocí metod konečných prvků, či rozdílů.

V mnoha případech při zavedení paralelizace dochází k vyřešení všech zmíněných problémů současně. Aby mohla být paralelizace zavedena, musí kód vykazovat určité vlastnosti. Mezi nejdůležitější vlastnosti patří – časté opakování výpočtu lišící se jen určitým parametrem a výsledek opakovaného výpočtu nezávisí na jiných výpočtech.

Paralelním počítáním na více-jádrových procesorech se zabývá např. pracoviště národního superpočítačového centra IT4Innovations na VŠB-TUO. Centrum se konkrétně zabývá metodami rozložení oblastí na metodě konečných a hraničních prvků a jejich použitím k řešení rozsáhlých inženýrských úloh.

Katedra stavební mechaniky na VŠB-TUO v průběhu posledních dvou let (2017-2018) řešila dva projekty zabývající se paralelními algoritmy založené na metodě konečných a hraničních prvků s názvy: *„Paralelizace výpočetních postupů ve stavební mechanice“* (Koktan, 2017) a *„Paralelní algoritmy pro analýzu desek na podloží“* (Koktan, 2018). Metoda konečných prvků obsahuje několik fází, kde je možné upravit algoritmus tak, aby využíval možnosti paralelního provádění operací na více procesorech. V rámci obou projektů byly řešené úlohy rozděleny na 2 základní fáze: paralelní výpočet jednotlivých podoblastí, tzn. především datový paralelismus a řešení úlohy na rozhraní (která může být také paralelizována). Výsledkem obou projektů je zrychlení výpočtu a příprava na možnost řešení rozsáhlých úloh na superpočítači.

Dalším příkladem paralelního počítání je zpracování mapových podkladů pro portál Mapy.cz, které proběhlo již v roce 2015, rovněž ve spolupráci se zmíněným pracovištěm IT4Innovations. Na základě dvourozměrných leteckých snímků měst a jejich blízkého okolí z různých úhlů pohledu byly vytvořeny 3D modely měst a jejich textury za použití výpočetně náročných algoritmů. Zpracované snímky pokrývají téměř 8000 km^2, odpovídající 1,5násobku rozlohy Moravskoslezského kraje. Paralelní zpracování 3D modelů měst výrazně zkrátilo dobu zpracování.

Typickým příkladem paralelismu, který řeší problémy spojené s velkým množství dat a výpočetní náročnosti je učení neuronové sítě. *„Neuronovou síť je možné definovat jako masivně paralelní procesor, který má sklon k uchovávání experimentálních znalostí a jejich dalšího využívání.“* (Olej, 2009) Neuronové sítě slouží především pro modelování, identifikaci a analýzu systémů, optimalizaci, predikci signálů (časových řad), klasifikaci atp. (Fiala, 2014)

Příkladem práce s neuronovou sítí je projekt s názvem: *„Analýza a zpracování rozsáhlých grafů“*, na jehož základě byla navržena implementace efektivních metod výpočtu spektrálního rozkladu matic (grafů) v oblastech spektrálního shlukování, kompresi grafů a implementace založená na paralelizaci výpočtů. V rámci projektu byly řešeny problémy spojené s rozsáhlostí dat a jejich nedostatečně rychlého zpracování. Výsledkem projektu je implementace jednotlivých metod, které umožňují zpracovávat rozsáhlé datové kolekce, paralelizovat výpočet a celkově tak zefektivnit výpočet. (Dvorský, 2012)

3 ANALÝZY VIDITELNOSTI

Analýza viditelnosti slouží pro určení viditelnosti z určitého místa. Cílem analýzy viditelnosti je vyhodnocení, zda pozorované místo je viditelné, či není viditelné z místa pozorování, neboť jej zakrývá překážka na terénu nebo samotný terén. Analýzy viditelnosti byly v minulosti využívané především pro vojenské účely. Jedno z prvních zdokumentovaných využití analýzy viditelnosti pochází z obléhání města Ath v Belgii v roce 1706. V dnešní době se analýzy viditelnosti používají v mnoha oblastech – územní plánování, architektura, komunikace, turismus apod. (Popelka, 2012)

Pro analýzu viditelnosti je důležité použití vhodných dat pro tvorbu podkladového povrchu. Analýza viditelnosti je zpravidla vyhodnocována na základě digitálního modelu terénu (DMT) nebo digitálního modelu povrchu (DMP). Digitální model terénu je na rozdíl od digitálního modelu povrchu zpravidla holý bez zahrnutí dalších výškových překážek (budov, vegetace, aj.). Proto je vhodnější digitální model povrchu, ve kterém jsou překážky zahrnuty. Analýzy viditelnosti se v GIS provádí téměř vždy nad povrchem reprezentovaným v rastrovém datovém typu.

Analýzy viditelnosti lze dělit na:

a. **Jednoduchou viditelnost** – která vyhodnocuje viditelnost pouze z jednoho místa pozorování. Výsledkem analýzy je rastr, jehož buňky jsou vyhodnocené jako neviditelné, pokud je jejich hodnota rovna 0 a viditelné, pokud se rovnají 1.

b. **Násobnou viditelnost** – která vyhodnocuje viditelnost z více pozorovacích míst. Buňky výstupního rastru nabývají hodnot 0 – neviditelné, nebo 1 – viditelné ze všech pozorovacích míst.

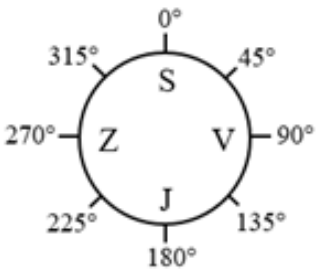
c. **Součtovou viditelnost** – která představuje algebraický součet dvou nebo více jednoduchých viditelností. Buňky výsledného rastru nabývají hodnot od 0 (neviditelné) až po hodnotu n (viditelné), kde n je počet pozorovacích míst. Hodnota buňky tak vypovídá o tom, z kolika pozorovacích bodů je viditelná, a tím se zvýrazní oblasti ve výsledném rastru, které jsou nejviditelnější ze zadaných pozorovacích míst.

d. **Totální viditelnost** – je speciálním případem součtové viditelnosti, u které jsou pozorovacími body všechny buňky studované oblasti. Viditelnost je tedy počítána jako součet jednoduchých viditelností ze všech možných pozorovacích bodů v dané oblasti. Čím vyšší hodnotu má výstupní buňka rastru, tím je buňka viditelnější v rámci celé testované oblasti.

e. **Pravděpodobnostní viditelnost** – která vyhodnocuje souhrnnou viditelnost z N rastrů viditelností, z nichž výpočtem aritmetického průměru pro každou buňku, získá každá buňka hodnotu z intervalu <0,1>. Hodnota blízká k 0 znamená, že buňka je téměř neviditelná a hodnota blízká k 1 signalizuje, že je pravděpodobně viditelná.

Výpočet analýzy viditelnosti vyžaduje dvě základní vstupní sady/vrstvy – digitální model povrchu a místo pozorování. Místo pozorování je zadáváno jako bod nebo buňka a jeho výška bývá interpolována z digitálního modelu povrchu, nebo bývá nadsazena do určité výšky nad povrchem. Díky tomuto, ale i z podstaty vyhodnocování viditelnosti se musí místo pozorování nacházet uvnitř oblasti pokryté DMP. Např. software ArcGIS má ve standardním nastavení výšku 1 m nad použitým výškovým modelem. Na výsledek výpočtu viditelnosti má zásadní vliv použitý algoritmus, a zahrnutí dalších vstupních parametrů do analýzy viditelnosti, které závisí na použitém softwaru. Vstupními parametry se rozumí parametry řídící analýzu viditelnosti především ve smyslu omezení hodnocené oblasti. Např. software ArcGIS umožňuje zahrnout do výpočtu následující parametry (Esri, 2016):

1. *SPOT* – slouží k určení nadmořské výšky pozorovacích bodů.
2. *Offset* – je parametr vertikálního nadsazení nad terénem skládající se z parametrů: OFFSETA (vyvýšení místa pozorovatele nad terénem) a OFFSETB (vyvýšení cíle pozorování nad terénem).
3. *Azimuth* – je parametr, který určuje horizontální úhel pohledu. Hodnoty úhlu jsou udávány ve stupních 0° až 360°, přičemž úhel 0° je orientován na sever. Záběr pohledu se většinou zadává výsečí mezi dvěma úhly – např. pro směr na jih (180°) je možné zadávat výseč 135°-225°.



Obr. 10 Výchozí kružnice parametru *Azimuth*

4. *Vertical angle* – je parametr, který určuje vertikální úhel pohledu. Úhly jsou vyjádřeny ve stupních v rozmezí 90°až -90°, kde úhel 0° představuje horizontální rovinu. Výseč -90°-90° reprezentuje viditelnost bez omezení. (Tzn. že je možné se dívat nejen před sebe, ale i nahoru a dolů – nad sebe, pod sebe.)

5. *Radius* – je parametr specifikující nejkratší pohledovou vzdálenost, tzn. že buňky, které jsou nad určitou (limitní) vzdáleností, mohou být z analýzy vyloučeny. Radius se používá při snížené dohlednosti vlivem meteorologických podmínek (např. mlha).

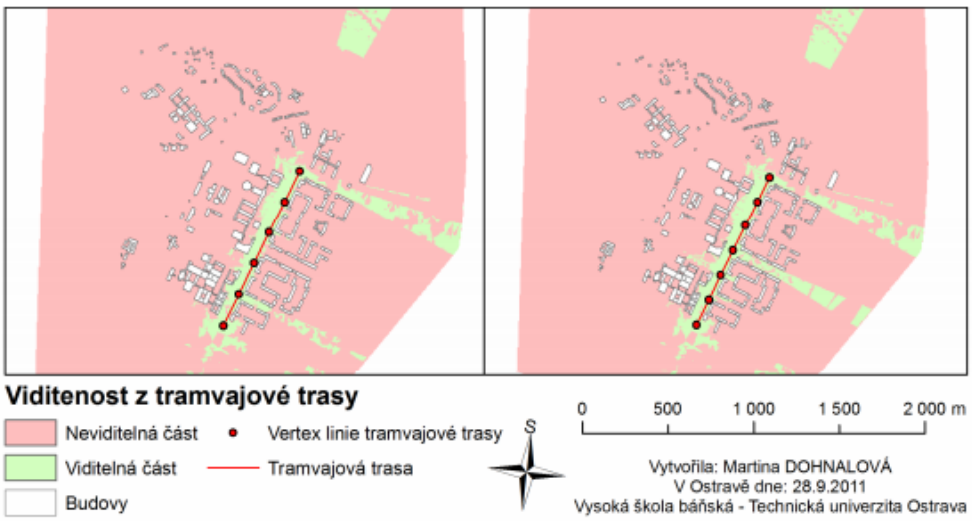
Mimo softwaru ArcGIS existuje mnoho dalších softwarů, které umožňují výpočet analýzy viditelnosti (např. GRASS GIS, IDRISI, SAGA, ERDAS aj.), a poskytují řadu nástrojů, které jsou vhodné pro jeho vyhodnocení.

3.1 Náročnost výpočtů viditelnosti

Analýza viditelnosti je citlivá na kvalitu a přesnost vstupních dat. Na požadavek přesnosti dat se váže problém náročnosti výpočtu analýzy viditelnosti. Náročnost výpočtu nastává zejména v případě výpočtu analýzy viditelnosti s velkou rozlohou terénu a jejím rozlišením (detailnější model terénu), větším počtem pozorovaných míst, ze kterých se viditelnost vyhodnocuje a větším počtem opakování výpočtu při stochastickém vyhodnocení analýzy (např. propagace potenciálních chyb terénu do výstupu analýzy). Náročnost výpočtu se nejčastěji týká následujících úloh:

- **Viditelnost z linie** (součtová či násobná analýza viditelnosti) – úloha viditelnosti z linie spočívá ve vyhodnocení jednoduché viditelnosti z liniového prvku, tzn. ze všech jeho uzlů a následným sloučením výsledků do jedné vrstvy. Souhrnně je vyhodnocována jako součtová či násobná viditelnost z těchto uzlů.

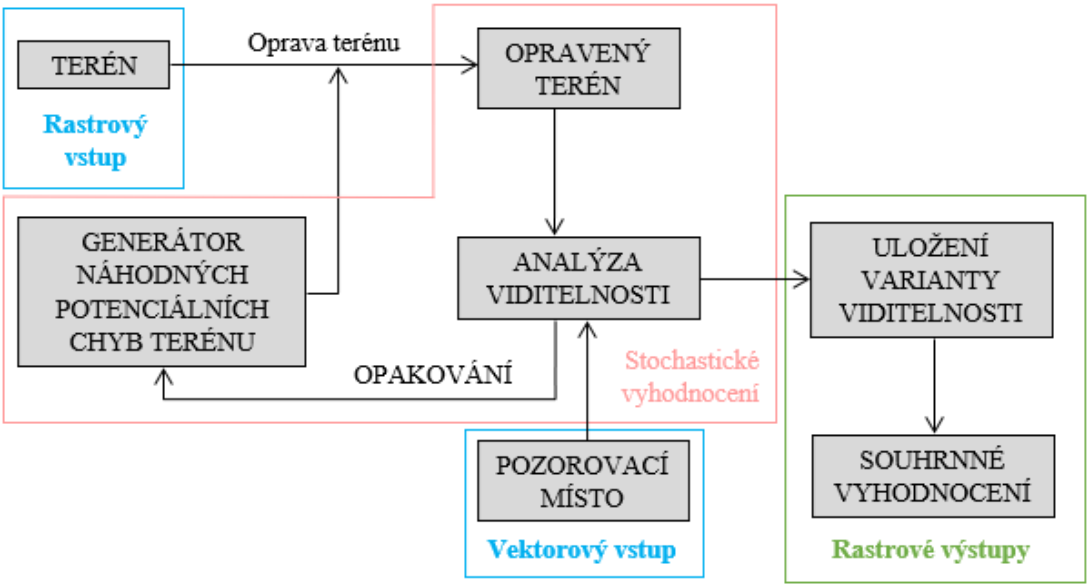
  Problémem úlohy viditelnosti z linie je velké množství míst pozorování. Tímto se zabývá bakalářská práce Pokročilé vyhodnocení viditelnosti z linie (viz kap. 4.1) a diplomová práce Citlivostní analýza vyhodnocení viditelnosti z linie, jejíž autorkou je M. Dohnalová (2012).



Obr. 11 Výsledky analýzy viditelnosti z tramvajové trasy (Dohnalová. 2012)

– **Totální viditelnost** – problémem úlohy totální viditelnosti je taktéž velké množství míst pozorování. Úlohou totální viditelnosti se např. zabývá dizertační práce Substanciálna analýza viditeľnosti v prostredí GIS od A. Rašové (2015).

– **Stochastické vyhodnocení viditelnosti** – stochastická simulace spočívá v několikanásobném modelování procesu náhodného charakteru. Simulováním náhody (viz obr. 12) lze vypracovat řadu variant řešení (viditelnosti).



Obr. 12 Stochastický model

Některé typy simulačních úloh mají specifické označení (např. Monte Carlo simulace, která se typicky využívá pro výpočet integrálů). Problémem úlohy stochastického vyhodnocení analýzy je mnoho opakování a zaobírá se jim dizertační práce Substanciálna analýza viditeľnosti v prostredí GIS od A. Rašové, 2015 (simulace Monte Carlo a Gaussovská geostatická simulace).

4 PRAKTICKÁ STUDIE

4.1 Paralelizované procesy

Byly vybrány následující úlohy:

- *viditelnost z linie* (viz kap. 3.1) – součtová viditelnost z linie,
- *stochastické vyhodnocení viditelnosti* (viz kap. 3.1) – pravděpodobnostní viditelnost.

První vybraná úloha byla řešena v rámci dříve zmíněné bakalářské práce (Lelek, 2016). Druhá vybraná úloha vyhází z obdobné úlohy, která byla řešena na prvním cvičení předmětu Modelování a simulace v geovědách v prvním ročníku navazujícího studia s názvem „Vliv nejistoty terénu na výsledky analýzy“ viz níže.

V rámci zmíněné bakalářské práce byl vytvořen nástroj, který slouží pro přípravu vstupní linie k následnému vyhodnocení viditelnosti. Vytvořený nástroj se skládá ze dvou submodelů tvořící hlavní model.

První submodel slouží k modifikaci linie (dělení linie na kratší úseky) a k výpočtu parametrů majících vliv na výsledky analýzy viditelnosti. Mezi faktory, které mají zásadní vliv na analýzu viditelnosti patří počet a rozmístění vertexů na vstupní linii. Při větším počtu vertexů plocha viditelného území roste a při menším počtu vertexů jsou naopak reálně viditelné plochy vyhodnoceny jako neviditelné. Za předpokladu, že bylo použito větší rozlišení rastru a menší krok dělení linie platí, že narůstá jak výpočetní čas, tak i celková výpočetní náročnost.

Druhý submodel je jednodušší a přehlednější než předcházející submodel a slouží k vyhodnocení analýzy viditelnosti a následné reklasifikaci výstupního rastru.

Hlavní model s názvem *„Pokročilé vyhodnocení viditelnosti z linie“ s*louží k propojení dvou výše popsaných submodelů, a ve výsledku poskytuje pokročilé vyhodnocení viditelnosti z linie a následnou reklasifikaci výstupního rastru. (Lelek, 2016)

Druhá úloha řešená na cvičení předmětu Modelování a simulace v geovědách spočívala ve zhodnocení důsledku výskytu náhodných chyb v terénu na základě vygenerování nahodilých chyb a následného opravení DMT o tyto chyby. Nikdy ovšem nelze vygenerovat potenciální chyby tak, aby přesně odrážely velikost a prostorové rozlišení skutečných chyb. Dá se pouze přiblížit realitě tak, že se generování chyb, oprava terénu a následná analýza provede opakovaně a nejčetnější vyhodnocení analýzy bude možné považovat za to nejpravděpodobněji správné.

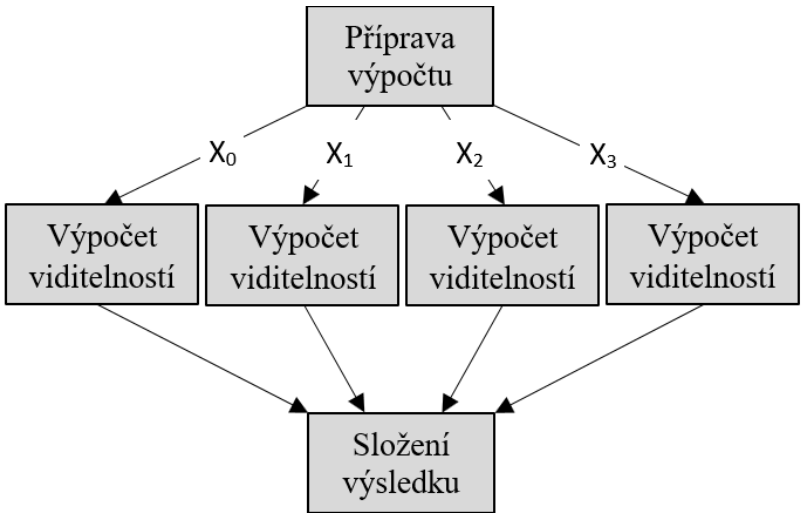
V rámci cvičení byla využita nejjednodušší varianta generování potenciálního rozložení chyb DMT (rovnoměrné rozdělení v daném rozmezí velikosti chyb). Po opravě terénu byla stochasticky vyhodnocována analýza stoku vody po terénu, která byla v rámci této diplomové práce nahrazena analýzou viditelnosti. Na základě stochastického vyhodnocení stoku vody po terénu byla vyhodnocena předpokládaná existence trvalého toku. (Růžičková, 2017)

4.2 Návrh způsobu paralelizace

Pro vybrané úlohy byla hledána vhodná forma vícevláknové paralelizace ze zmíněných metod v kap. 2.3, která by mohla být aplikována na obě úlohy. Výběr vhodné paralelizace byl řešen z pohledu možnosti rozdělení dat a rozdělení úloh, které se v rámci výpočtu používají. Výhodou vybrané formy vícevláknové paralelizace je, že běží nad operačním systémem a z tohoto důvodu je přenositelnější v rámci operačních systémů (MS Windows, GNU/Linux).

První zmíněnou metodou paralelizace ve více vláknech (viz kap. 2.3) je datová paralelizace, kterou lze využít na obě konstruované úlohy. Úlohu viditelnosti z linie lze paralelizovat dle metody datového paralelismu ve smyslu rozdělení výpočtu viditelnosti ze všech pozorovacích míst na linii do několika souborů, přičemž dojde k rozdělení linie na několik částí s menším počtem pozorovacích míst. Pro takto paralelizovanou úlohu by graf datové závislosti vypadal jako na obr. 13., kde příprava výpočtu spočívá právě v rozdělení míst pozorování do více souborů, DMT se nedělí, zůstává v celku.

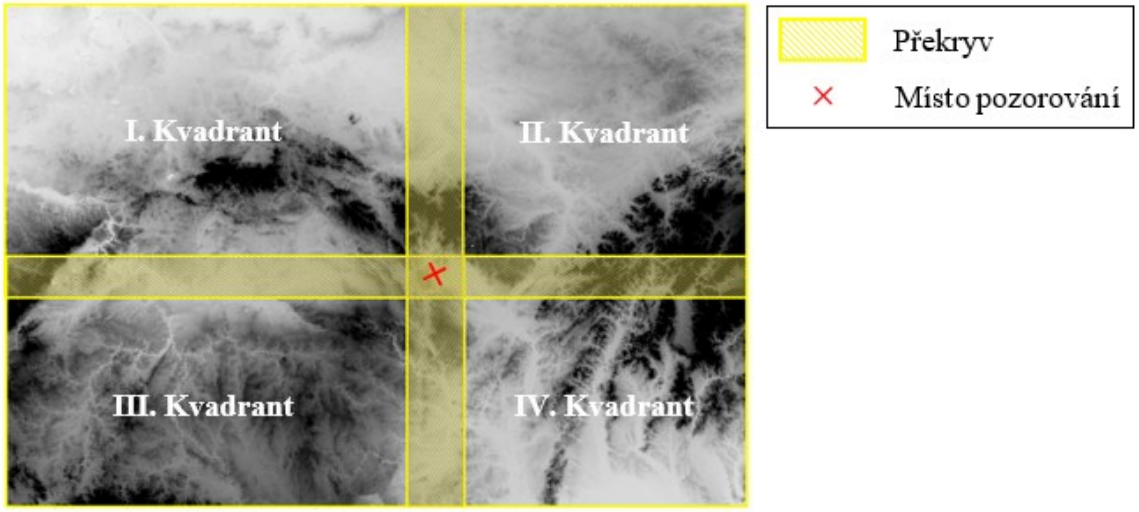
Nicméně i na straně DMT lze rozsah zpracování omezit, pokud bude počítáno s dohledností. Dohlednost je maximální vzdálenost, ze které jsme schopni okem rozeznat cíl pozorování. Na základě dosahu viditelnosti lze DMT ořezat kruhovým okolím s maximální vzdálenosti dohlednosti.



Obr. 13 Paralelní řešení úlohy viditelnosti z linie

V případě úlohy stochastického vyhodnocení viditelnosti by bylo možné paralelizaci provést obdobným způsobem jako v předchozí úloze. Stochastické vyhodnocení spočívá v několikanásobném opakování výpočtu, tudíž by bylo možné rozdělit opakující se výpočet viditelnosti do několika jednotlivých výpočtů. Např. pro výpočet viditelnosti opakovaný 500x by bylo možné rozdělit na výpočet všech simulovaných viditelností do 4 vláken po 125 opakováních. Tzn. že se nedělí vstupní datové sady, ani funkce výpočtu, ale dělí se celá opakování výpočtu s tím, že místo pozorování, ze kterého je prováděna analýza viditelnosti je jen jedno stále stejné. Graf datové závislosti by vypadal obdobně jako na předchozím obrázku.

Dalším možným řešením, jak rozdělit výpočet úlohy stochastického vyhodnocení viditelnosti, tak aby bylo možné použít metodu datového paralelismu, je rozdělit terén do jednotlivých kvadrantů. Výpočet by tak mohl být rozdělen do takového počtu vláken, do kolika kvadrantů byl terén rozdělen. Problémem této úlohy je překryv kvadrantů, neboť pozorovací místo, ze kterého je prováděna analýza viditelnosti, se musí nacházet ve všech kvadrantech, do kterých byl terén rozdělen (viz obr. 14).



Obr. 14 Rozdělení terénu do 4 kvadrantů pro 4 vlákna

Metoda funkcionálního paralelismu a metoda pipelining, které byly taktéž popsány v kapitole 2.3, nelze aplikovat ani na jednu z řešených úloh, protože výpočet viditelnosti je v GIS jako jeden nástroj. Pro rozdělení by musel být nástroj přeprogramován. Samostatně vystupující operace, které lze oddělit jsou příprava dat (např. u stochastického vyhodnocení příprava DMT) pro výpočet viditelnosti a samostatný výpočet viditelnosti. Nicméně příprava dat a výpočet viditelnosti na sebe sekvenčně navazují, a z tohoto důvodu výpočet nelze paralelizovat. Proto zbylé zmíněné metody paralelismu nebyly zahrnuty do praktické studie.

4.3 Technické a programové řešení

Pro testování všech algoritmů bylo využito školních počítačů ve třech školních učebnách, které jsou k dispozici studentům oboru geoinformatiky. Algoritmy byly spouštěny na počítačích s následujícími konfiguracemi.

Tab. 1 Parametry využitých školních počítačů

Počítač	Procesor			Počet vláken	RAM	Disk
	Počet	Typ	Frekvence			
PCJ424K	6	Intel® Core™ i7-8700 CPU	3,19 GHz	12	16 GB	1843 GB
PCJ423V	4	Intel® Core™ i5-7500T CPU	2,71 GHz	4	8 GB	239 GB
PCJ339S	4	Intel® Core™ E5-1607 0 CPU	3,00 GHz	4	8 GB	465 GB

Příprava testovacích dat (viz kap. 4.4.1) byla provedena v programu ArcMap 10.6, který slouží pro mapové úlohy, prostorové analýzy a editaci dat. K přípravě dat byly využity například tyto programové funkce:

- Define Projection – pro definování požadovaného souřadnicového systému terénu,
- Append – pro spojení jednotlivých dlaždic terénu do jedné vrstvy,
- Buffer – pro vytvoření polygonové (ořezové) vrstvy kolem testované linie,

Po přípravě testovacích dat byly vytvořeny celkem 4 skripty. Dva sekvenční skripty pro obě zmiňované úlohy (viz kap. 4.4.2) a zbylé dva skripty pro jejich paralelní zpracování (viz kap. 4.4.3). Skripty byly sepsány v jazyce python, který je podporován stěžejním programem této práce GRASS GIS (7.4.1, 7.4.0, 7.2.1), ve kterém byly skripty spouštěny a počítány. Všechny verze programu podporovaly naprogramovanou analýzu viditelnosti, tudíž nemusela být upravována pro jednotlivé verze programu.

Geografický informační systém GRASS (Geographic Resources Analysis Support System) je svobodný Open Source Software, který umožňuje správu geografických 2D/3D rastrových a vektorových dat, obrazových záznamů, produkci grafických výstupů, časoprostorové modelování a vizualizaci mnoha typů dat na platformách – GNU/Linux, MS Windows, MAC OS X, aj.). Mimo jiné umožňuje provádět paralelní výpočty.

GRASS GIS obsahuje přes 350 modulů pro vykreslování map, zpracování multispektrálních obrazových dat, manipulaci s rastrovými a vektorovými daty, vytváření, spravování a ukládání prostorových dat. Několik málo modulů je implementováno v programovacím jazyce C++, jiné jsou k dispozici v podobě skriptů v jazyce Python. (Portál FreeGIS, 2015). Moduly, které byly využity při programování obou úloh, jsou popsány níže v kap. 4.4.2 a 4.4.3.

4.4 Testování efektivity paralelizace

Efektivita paralelizace byla hodnocena na základě vybraných hodnotících parametrů, jež jsou popsány v kap. 2.1. Prvním hodnotícím parametrem byla doba běhu výpočtu (výpočtu bez paralelizace T_S a výpočtu s paralelizací T_P), na jejímž základě byl vyhodnocen poměr času f stráveného výpočtem sekvenční části ku celkovému času. Následně dle poměru bylo vyhodnoceno dosažené zrychlení S, které vychází z Amdahlova zákona (viz kap. 2).

Další parametry, dle kterých byla hodnocena efektivita paralelizace, jsou:

- paralelní efektivita E,
- paralelní cena C,
- paralelní režie H.

Některé vybrané hodnotící parametry byly vyhodnoceny nejen matematicky ale i graficky. Všechny výsledky jsou vizualizovány v kap. 4.4.4.

4.4.1 Testovací data

Oba algoritmy byly vyhodnoceny na základě DMT pro Českou republiku a linie pozorování představující železniční trať vedoucí z Bohumína do Prahy.

Digitální výškový model

Digitální výškový model poskytuje data výšek terénu založené na digitálním modelu terénu. Pro analýzu viditelnosti v obou dvou úlohách byl využit jako podklad digitální výškový model Evropy (EU-DEM). Vzhledem k tomu, že žádný jednotný zdroj dat neposkytuje konzistentní a úplné celoevropské pokrytí, byl pořízen digitální výškový model Evropy v rámci projektu GMES RDA (Copernicus programme). Jedná se o hybridní produkt, který je založený na údajích ze SRTM a ASTER GDEM, ale také s využitím údajů o výšce z volně dostupných ruských topografických map.

Pomocí digitálního výškového modelu Evropy lze modelovat povodně a řídit povodňová rizika. Dále lze digitální výškový model využít pro analýzu řek (přítoky, povodí, odvodňovací síť), nebo pro studii viditelnosti či pokrytí. Digitální výškový model Evropy je poskytován v souřadnicovém systému ETRS89-LAEA (EPSG kód 3035)

s rozlišením 25 m a vertikální přesností +/- 2,31 m RMSE pro Českou republiku viz obr. 15.

	n	Mean error (m)	St.dev. (m)	RMSE (m)	LE95 (m)
Albania	1937	-1,76	1,68	2,44	4,77
Andorra	1*	7,41	n.a.	7,41	14,53
Austria	4757	-1,84	1,84	2,60	5,09
Belgium	6567	-0,45	1,51	1,58	3,09
Bosnia and Herzegovina	2665	-1,88	1,90	2,68	5,25
Bulgaria	16263	-0,20	1,78	1,79	3,51
Canary Islands	56*	1,57	2,41	2,86	5,60
Croatia	9953	-1,61	2,07	2,62	5,13
Cyprus	3980	0,27	1,51	1,54	3,01
Czech Republic	8803	-1,04	2,07	2,31	4,53
...	...	...	...	...	...

Obr. 15 Výsledky základních přesností EU-DEM (Dufourmont, et al., 2014)

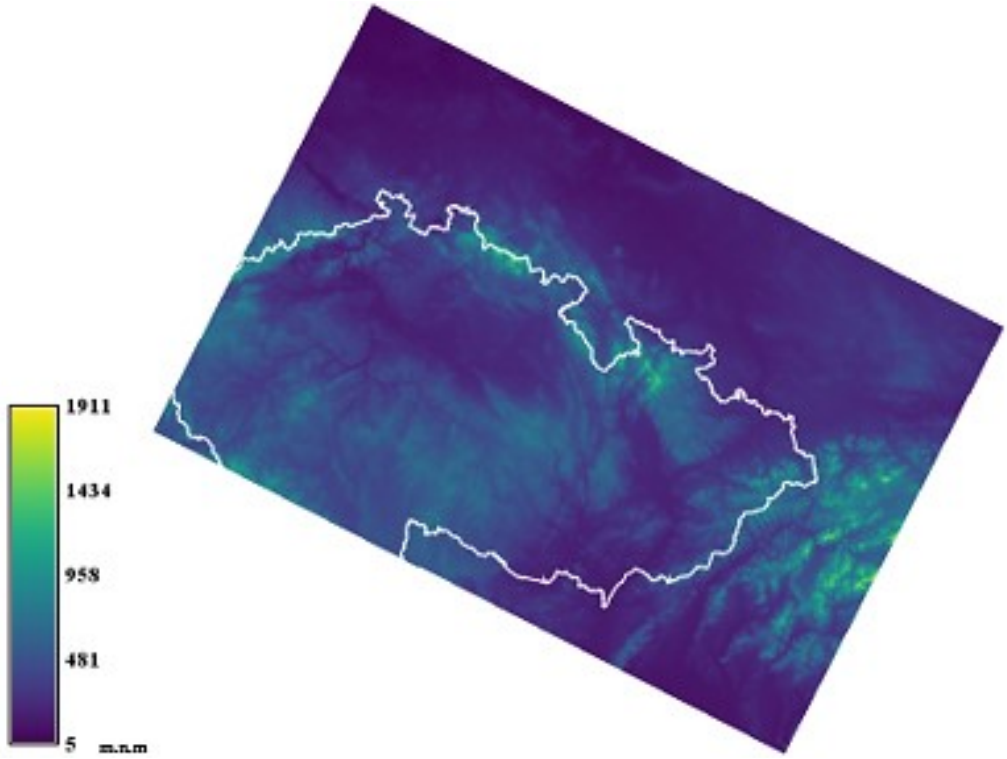
Souvislý datový soubor je rozdělen do jednotlivých dlaždic o velikosti 1000x1000 km. Jednotlivé dlaždice lze bezplatně stáhnout z internetových stránek – https://www.eea.europa.eu/data-and-maps/data/eu-dem. Pro analýzu viditelnosti byl vytvořen podklad z následujících 4 dlaždic:

- EUD_CP-DEMS_4500025000-AA.tif,
- EUD_CP-DEMS_4500035000-AA.tif,
- EUD_CP-DEMS_5500025000-AA.tif,
- EUD_CP-DEMS_5500035000-AA.tif.

Podklad ze zmíněných dlaždic digitálního výškového modelu Evropy byl vytvořen v programu ArcMap, neboť práce s tak velkým souborem vykazovala jisté problémy při transformaci souřadnicového systému v programu GRASS GIS, který je stěžejní v rámci této práce. Z tohoto důvodu byly dlaždice digitálního výškového modelu Evropy naimportovány do programu ArcMap, jejich souřadnicový systém byl transformován do souřadnicového systému testované linie (S-JTSK / Krovak East North) a následně byly spojeny do jedné vrstvy (teren.tif). Takto upravený terén vstupuje do výpočtu obou úloh.

Dále již v rámci výpočtu je vrstva terénu ořezána bufferem o velikosti 10 000 m, který byl vytvořen kolem testovací linie z důvodu dohlednosti, protože nemá smysl

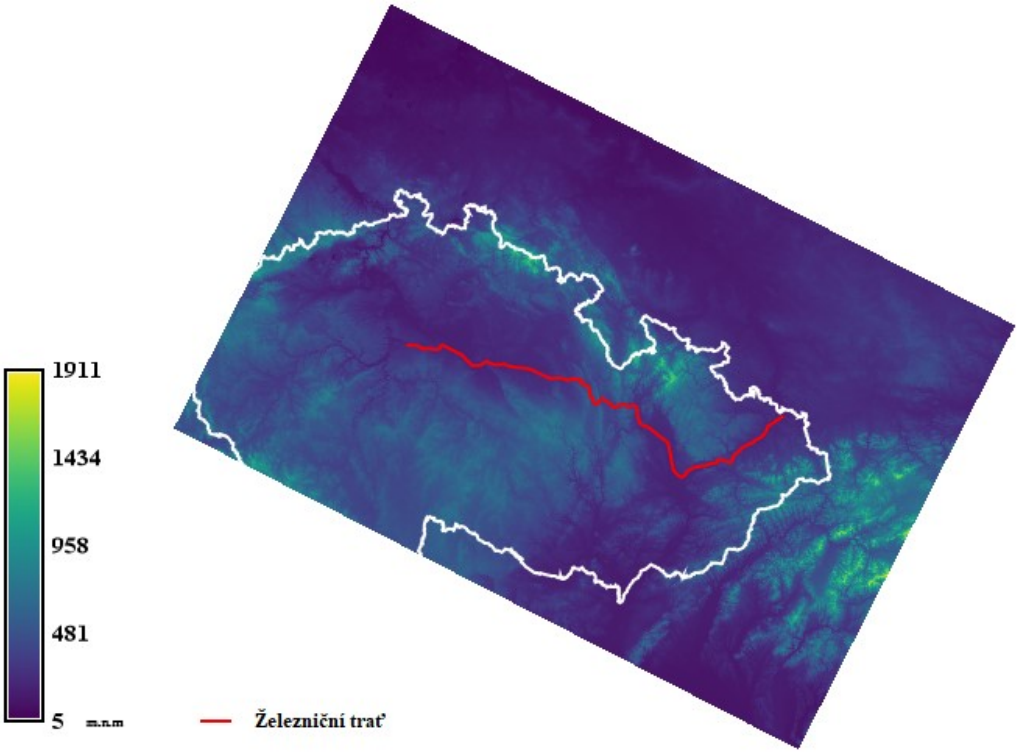
vyhodnocovat viditelnost ve větších vzdálenostech. Dohlednost je velice proměnná a závisí na nadmořské výšce. Např. je poměrně malá v nížinách a středních polohách, naopak ve výšce nad 1000 m je většinou o dost větší. Dohlednost se většinou vyjadřuje v metrech, popř. v kilometrech, přičemž do 5 km se přesnost udává po 100 metrech, do 30 km potom po 1 km a od 30 km po 5 km. Za výbornou dohlednost se v meteorologii označuje taková dohlednost, která přesahuje přes 50 km. Nicméně nad 70 km se dohlednost v meteorologii už přesněji neurčuje. (Žák, 2016)



Obr. 16 Upravený terén

Testovaná linie

Železniční trať byla upravena z volně dostupné vrstvy železniční síť z datové sady ArcČR® 500, která obsahuje mimo železniční tratě i jiné vektorové vrstvy, např. vrstvu železniční stanice, která byla využita při výpočtu druhé úlohy, kde do výpočtu viditelnosti vstupuje jedna vybraná železniční stanice pro opakované vyhodnocení viditelností. Všechny dílčí linie zvolené trasy byly spojeny do jedné souvislé linie s počátkem a koncem pro snazší úpravu linie pozorování v programu GRASS GIS, ve kterém byly analýzy viditelnosti provedeny.



Obr. 17 Linie pozorování

4.4.2 Výpočet bez paralelizace

Oba výpočty bez paralelizace pracují s knihovnou skriptů GRASS Python, která se importuje příkazem – *import grass.script as g.script*. Z této knihovny byly využity následující moduly:

- *OS* – poskytuje funkce závislé na operačním systému. Modul zejména slouží pro operace s jednotlivými soubory. Poskytuje např. funkce pro čtení a zápis souborů, pro manipulaci s cestami k datům apod.
- *SHUTIL* – poskytuje řadu operací na úrovni souborů a kolekcí souborů. Poskytuje zejména funkce, které podporují kopírování a odstraňování souborů.
- *DATETIME.DATETIME* – kombinuje datum a čas. Vrací atributy – rok, měsíc, den, hodina, minuta, sekunda a mikrosekunda.

Algoritmy se skládají z načtení vstupních dat, jejich úpravy lišící se dle zpracovávané úlohy, výpočtu a časového vyhodnocení úlohy.

Viditelnost z linie

Cílem sekvenční úlohy viditelnosti z linie je vyhodnotit časovou náročnost analýzy viditelnosti ze všech uzlů testované linie, které byly vytvořeny v rámci přípravy výpočtu viditelnosti viz níže.

Vstupními daty úlohy viditelnosti z linie jsou – linie pozorování odpovídající železniční trati vedoucí z Bohumína do Prahy (viz kap. 4.4.1) a upravený digitální výškový model (viz kap. 4.4.1), který sloužil jako podklad pro vyhodnocení analýzy viditelnosti. Pro obě vstupní vrstvy (vektorová – linie pozorování a rastrová – terén) byl nadefinován import do prostředí GRASS GISu –

```
def importVectorData():
    input_vector = os.path.join(PROJECT_ABS_PATH, SHP_FILE_PATH)
    name = os.path.splitext(os.path.basename(SHP_FILE_PATH))[0]
    gscript.run_command('v.in.ogr', input=input_vector, layer=name,
output=name, overwrite=True, flags='o')
    return name
def importDataIntoMapProcess(filePath):
    input_raster_data = os.path.join(PROJECT_ABS_PATH, filePath)
    name = os.path.splitext(os.path.basename(filePath))[0]
    gscript.run_command('r.in.gdal', input=input_raster_data,
output=name, flags='o', overwrite=True)
    return name
```

Vektorová vrstva je importována do prostředí GRASS GISu na základě nastavené cesty k požadované vektorové vrstvě (SHP_FILE_PATH) prostřednictvím funkce *v.in.ogr* z knihovny OGR příkazem *gscript.run_command*, který volá příkaz z GRASSu.

Rastrová vrstva je importována do GRASS GISu obdobným způsobem s tím rozdílem, že příkazem *gscript.run_command* je volána funkce *r.in.gdal*, která importuje rastrová pomocí knihovny GDAL.

Po importu vstupních vrstev byl nastaven region na DMT pomocí funkce *g.region*, která spravuje hranice pro zpracovávanou oblast.

```
def setRegionToRasterMap(rasterMap):
    print('Set region to match raster map
<{}>'.format(','.join(rasterMap)))
    gscript.run_command('g.region', raster=rasterMap)
```

Aby mohla být provedena analýza viditelnosti, musela být liniová vrstva dále upravena. Úprava zahrnovala tvorbu pravidel, rozdělení linie pozorování a převod na ASCII. Vytvořená pravidla sloužila pro rozdělení linie pozorování na jednotlivé segmenty (vertexy). Pravidla byla vytvořena na základě popisu v grass manuálu (viz obr. 18) a uložena do .txt souboru.

```
P       1       1       0
P       2       1       1000
P       3       1       2000
P       4       1       3000
P       5       1       4000
P       6       1       5000
...
P       350     1       349000
```

Obr. 18 Příklad z vytvořených pravidel

Písmeno P v prvním sloupci znamená, že je požadována nová bodová vrstva. V případě liniové vrstvy by bylo písmeno P nahrazeno písmenem L. Druhý sloupec označuje id jednotlivých bodů. Třetí sloupec znamená do kolika kategorií budou nově vytvořené body rozděleny. A poslední sloupec znamená, v jaké vzdálenosti jsou jednotlivé body vytvořeny. V tomto případě jsou body rozděleny do jedné kategorie ve vzdálenosti 1 km. Na základě těchto pravidel bude vytvořeno 349 bodů na 349 km dlouhé železniční trati.

Následně byla vytvořena bodová vrstva dle výše uvedených pravidel pomocí funkce *v.segment*, do které vstupuje vektorová vrstva linie pozorování a vytvořená pravidla, dle kterých je linie pozorování rozdělena na jednotlivé segmenty.

```
def createPoints(vectorData):
    name = vectorData + '_seg'
    print('Creates points/segments from vector lines and pravidla
<{}@{}>'.format(name, MAPSET))
    input_vector_map = '{}@{}'.format(vectorData, MAPSET)
    pravidla = os.path.join(PROJECT_ABS_PATH, pravidla_REL_PATH)
    gscript.run_command('v.segment', input=input_vector_map,
                        output=name, rules=pravidla, overwrite=True)
    return name
```

Poté byla bodová vrstva převedena do formátu GRASS ASCII prostřednictvím funkce *v.out.ascii*, pomocí které lze převézt body, linie, hranice, ad., přičemž defaultně je nastaven převod vrstvy typu bod. Výsledkem převodu je vrstva obsahující souřadnice všech segmentů (míst pozorování), ze kterých bude provedena analýza viditelnosti.

Po úpravě liniové vrstvy následovala úprava velikosti vstupního terénu z důvodu snížení výpočetní, časové i paměťové náročnosti během výpočtu viditelnosti. Úprava velikosti terénu byla provedena v následujících 3 krocích:

1. Vytvoření obalové vrstvy tzv. bufferu kolem všech uzlů železniční tratě pomocí funkce *v.buffer*, o velikosti obalové vrstvy 10 000 m, na základě dohlednosti (viz kap. 4.4.1). Obalová vrstva je defaultně tvořena kolem vstupní vrstvy typu bod, proto není nutné ve výpočtu uvádět typ vstupní vrstvy.

```
def createBuffer(vectorData):
    name = vectorData + '_buffer'
    print('Creates a buffer around vector features of given
type <{}@{}>'.format(name, MAPSET))
    input_vector_map = '{}@{}'.format(vectorData, MAPSET)
    gscript.run_command('v.buffer', input=input_vector_map,
                        output=name, distance=BUFFER_DISTANCE,
                        overwrite=True)
    return name
```

2. Převedení obalové vrstvy na rastr, pomocí funkce *v.to.rast*, aby mohl být touto vrstvou ořezán terén.

```
def convertToRasterMap(vectorData, useVal):
    input_vector_map = '{}@{}'.format(vectorData, MAPSET)
    name = vectorData + '_rast'
    gscript.run_command('v.to.rast', input=input_vector_map,
                        output=name, use=useVal,
                        overwrite=True)
    return name
```

3. Ořezání terénu obalovou rastrovou vrstvou pomocí rastrového kalkulátoru. Funkce rastrového kalkulátoru je volána příkazem *r.mapcalc*. Na základě této funkce vzniká nová rastrová vrstva.

```
def rasterMapCalc(mapLayer, bufferRasterMap):
    name = '{}_buffer'.format(mapLayer)
    expression = '{} = {}@{} - {}@{}'.format(name, mapLayer,
                                MAPSET, bufferRasterMap, MAPSET)
    print('Raster map calculator <{}>'.format(expression))
    gscript.run_command('r.mapcalc', expression=expression,
                        overwrite=True)
    return name
```

Takto upravená linie i terén vstupují do analýzy viditelnosti. Výpočet viditelnosti byl proveden sekvenčně pro všechny uzly linie pomocí funkce *r.viewshed*. Funkce používá pro určení viditelnosti následující model – předpokládá se, že výška buňky je proměnná, a že je interpolována skutečná výška bodu spadajícího do buňky, nikoli však identického centra buněk. Terén je tedy vnímán jako hladký povrch. Dva body jsou vzájemně viditelné, pokud jejich přímka není protkána terénem. Výška pro libovolný bod x v terénu je interpolována ze čtyř okolních sousedů. To znamená, že tento model provádí bilineární interpolaci výšek.

Tento model je vhodný jak pro rastry s nízkým i vysokým rozlišením, tak pro terén s plochými a strmými svahy.

Parametry funkce *r.viewshed* ve výpočtu byly ponechány defaultně nastavené mimo výšky pozorovatele, která byla nastavena na 2,5 m.

```
def viewshedAnalysis(vectorData, mapLayer, coordinatesList):
    print('Computes the viewshed of a point on an elevation raster map.')
    i = 0
    viewshed_maps = []
    input_map_layer = '{}@{}'.format(mapLayer, MAPSET)
    for coordinates in coordinatesList:
        start_time = datetime.now()
        i += 1
        name = vectorData + '_viewshed_' + str(i)
        print('Viewshed <{}@{}>'.format(name, MAPSET))
        try:
            gscript.run_command('r.viewshed', input=input_map_layer,
                                output=name, memory=MEMORY,
                coordinates=(coordinates[0], coordinates[1]),
                                observer_elevation=5.0)
            viewshed_maps.append(name)
        except:
            pass
        writeTimelaps('{} | {} | {}'.format(coordinates[0],
                    coordinates[1], datetime.now()- start_time))
    return viewshed_maps
```

Skript analýzy viditelnosti prochází všechny body (řádky) v ASCII souboru, vytáhne si jeho souřadnice a použije zmíněný nástroj *r.viewshed* pro výpočet. Po provedení analýzy viditelnosti ze všech segmentů linie pozorování je volána procedura *totalviewshed*, která souhrnně vyhodnotí výsledky viditelností, sloučí je do jedné výsledné vrstvy pomocí rastrového kalkulátoru *r.mapcalc*.

```
def totalviewshed(vectorData, viewshedMaps):
    name = vectorData + '_totalviewshed'
    expression = ''
    for viewshedMap in viewshedMaps:
        if expression == '':
            expression = '{} = {}@{}'.format(name, viewshedMap, MAPSET)
        else:
            expression = '{} + {}@{}'.format(expression, viewshedMap, MAPSET)
    rasterMapCalcNew(expression)
    return

def rasterMapCalcNew(expression):
    print('Raster map calculator <{}>'.format(expression))
    gscript.run_command('r.mapcalc', expression=expression, overwrite=True)
    return
```

Výsledný čas, který byl spotřebován pro výpočet analýzy viditelnosti, je zapisován do .txt souboru v následující podobě:

```
...
-727730.24012746 | -1045337.54872221 | 0:03:22.232000
-728729.86865021 | -1045310.29406194 | 0:03:34.269000
-729729.49717296 | -1045283.03940167 | 0:03:23.486000
-730719.51718797 | -1045206.05397454 | 0:03:24.020000
--- Elapsed Time 21:33:13.937000 ---
```

Obr. 19 Výsledek časové náročnosti výpočtu

Celý skript zpracované sekvenční úlohy viditelnosti z linie je k dispozici viz příloha č. 1. Výsledky byly získány spuštěním vytvořeného skriptu přes konzoli příkazem – *python *.py* v programu GRASS GIS na jednotlivých počítačích. Získané výsledky jsou vizualizovány a porovnány v kapitole 2.4.4

Stochastické vyhodnocení viditelnosti

Cílem sekvenční úlohy stochastického vyhodnocení viditelnosti je vyhodnotit časovou náročnost výpočtu analýzy viditelnosti, která je provedena opakovaně z jednoho místa pozorování.

Vstupními daty úlohy stochastické vyhodnocení viditelnosti jsou – vybraná železniční stanice „Zábřeh na Moravě“ a upravený digitální výškový model (viz kap. 2.4.1), který opět sloužil jako podklad pro vyhodnocení analýzy viditelnosti. Pro obě vstupní vrstvy byl nadefinován import do prostředí GRASS GISu stejně jako v předchozí úloze pomocí funkce *v.in.ogr* (pro vektorovou vrstvu) a funkce *r.in.gdal* (pro rastrovou vrstvu).

Po importu vstupních vrstev byl nastaven region na DMT pomocí funkce *g.region*, aby mohl být terén dále upravován. Následně byl terén upraven obdobným způsobem jako v předchozí úloze. Kolem vybrané železniční stanice byla vytvořena obalová vrstva o velikosti 10 000 m, na základě zvolené dohlednosti, prostřednictví funkce *v.buffer*. Tato vrstva byla poté převedena na rastr pomocí funkce *v.to.rast*, aby mohl být novou rastrovou vrstvou ořezán stávající terén prostřednictvím rastrového kalkulátoru *r.mapcalc*. Po úpravě terénu byla vektorová vrstva železniční stanice převedena do formátu ASCII GRASS funkcí *v.out.ascii*, pro následující stochastické vyhodnocení viditelnosti.

Stochastické vyhodnocení viditelnosti metodou Monte Carlo spočívá v několikanásobném opakování stochastického výpočtu viditelnosti, který byl v případě této úlohy

opakován 400x. Vyhodnocení stochastického výpočtu viditelnosti zahrnovalo následující kroky:

- Vytvoření prázdné rastrové vrstvy pro ukládání jednotlivých variant výsledků viditelnosti (před cyklem).

```
freqOfTerrain =
produceRasterMapOfRandDeviates('freqOfTerrain', 0, 0)
```

- Generování rastru náhodných chyb s rozptylem hodnot +-2,31 m dle EU-DEM viz kap. 4.4.1. (v cyklu)

- Sečtení terénu s rastrem náhodných chyb z důvodu opravy terénu vlivem možného výskytu náhodných chyb. (v cyklu)

```
tmpVarOfViewshed = ''
for i in range(0, 400):
    mistakes = produceRasterMapOfRandDeviates
        ('mistakes', -2.31, 2.31)
    expression = '{} = {}@{} + {}@{}'.format
        (terrainModification, bufferMapLayer,MAPSET,
         mistakes, MAPSET)
    rasterMapCalcNew(expression)
    viewshed_map = viewshedAnalysis
        (vectorData, terrainModification, coordinates, i)
    tmpVarOfViewshed = 'varOfViewshed_' + str(i)
    expression = '{} = {}@{} + {}@{}'.format
        (tmpVarOfViewshed, varOfViewshed, MAPSET,viewshed_map,
         MAPSET)
    rasterMapCalcNew(expression)
    varOfViewshed = tmpVarOfViewshed
```

- Vyhodnocení viditelnosti na základě opraveného terénu. (v cyklu) Samotný výpočet viditelnosti je proveden stejně jako v první úloze prostřednictvím funkce *r.viewshed* s defaultně nastavenými parametry mimo výšky pozorovatele, která byla nastavena na 2,5 m. Slouží k tomu samostatná procedura viewshedAnalysis, která je volána v předchozím kódu.

```
def viewshedAnalysis(mapLayer, i):
    print('Computes the viewshed of a point on an elevation
           raster map.')
    viewshed_map = VECTOR_DATA + '_viewshed_thread{}_{}'.format
          (str(THREAD_NUM), str(i))
    input_map_layer = '{}@{}'.format(mapLayer, MAPSET
    start_time = datetime.now()
    printSync(thread_num, 'Viewshed <{}@{}>'.format
    viewshed_map, MAPSET))
    try:
    gscript.run_command('r.viewshed', input=input_map_layer,
                        output=viewshed_map, overwrite=True,
                        coordinates=(coordinates[0],coordinates[1]),
                        observer_elevation=2.5, flags='b')
    except:
```

```
            pass
        writeTimelaps('{} | {} | {}'.format
           (coordinates[0], coordinates[1],datetime.now() - start_time))
        return viewshed_map
```

Po provedení opakované analýzy viditelnosti z jednoho místa pozorování jsou výsledky souhrnně vyhodnoceny, sloučeny do jedné výsledné vrstvy prostřednictvím funkce *r.mapcalc*. Slouží k tomu samostatná procedura rasterMapCalcNew, která je také v předchozím kódu (stejně jako procedura viewshedAnalysis) volána.

```
def rasterMapCalcNew(expression):
    print('Raster map calculator <{}>'.format(expression))
    gscript.run_command('r.mapcalc', expression=expression,
                        overwrite=True)
    return
```

Celý skript zpracované sekvenční úlohy stochastického vyhodnocení viditelnosti je k dispozici viz příloha č. 2. Získané výsledky stochastického vyhodnocení viditelnosti na jednotlivých počítačích jsou vizualizovány a porovnány v kapitole 4.4.4.

4.4.3 Výpočet s paralelizací

Výpočty s paralelizací pracují mimo zmíněných modulů (viz předchozí kapitola) také s moduly:

- *SYS* – který poskytuje řadu funkcí a proměnných, které lze použít k manipulaci s různými částmi běhového prostředí Pythonu.
- *THREADING* – staví na nízko-úrovňových vlastnostech vlákna, což usnadňuje práci s vlákny. Použití vláken tak umožňuje programu spouštět více operací současně ve stejném procesu.

Viditelnost z linie

Cílem paralelní úlohy viditelnosti z linie je dosažení lepších výsledků v porovnání se získanými výsledky úlohy viditelnosti z linie bez paralelizace.

Sekvenční část výpočtu úlohy zahrnuje veškeré předzpracování vstupujících vrstev do výpočtu viditelnosti. Tato část je shodná s částí předešlého výpočtu úlohy bez paralelizace. V rámci sekvenčního skriptu je definován import vstupních vrstev – linie pozorování (železniční tratě) a digitálního výškového modelu (terénu). Po importu vstupních vrstev byl nastaven region na DMT prostřednictvím funkce *g.region* a následně upravena linie pozorování. Linie pozorování je v rámci úpravy rozdělena na jednotlivé

segmenty (uzly) dle zmíněných pravidel (viz kap. 4.4.2) pomocí funkce *v.segment*. Bodová vrstva segmentů byla poté převedena do formátu GRASS ASCII funkcí *v.out.ascii* pro následné paralelní vyhodnocení viditelnosti ze všech uzlů testované linie.

Před paralelním zpracováním výpočtu viditelnosti byla dále upravena velikost terénu. Kolem linie pozorování byla vytvořena obalová vrstva (buffer) o velikosti 10 000 m prostřednictvím funkce *v.buffer*, stejně jako v první úloze, a převedena do rastrového formátu funkcí *v.to.rast*. Následně byl touto rastrovou vrstvou ořezán terén prostřednictvím rastrového kalkulátoru *r.mapcalc*.

Výpočet viditelnosti je již proveden paralelně ve 4 vláknech. Nejjednodušším způsobem, jak využít vlákna pro výpočet, je vytvořit instanci pomocí funkce *target* a zavolat metodu *start ()*, aby mohla začít vlákna pracovat.

```
def startThreads(vectorData, mapLayer, coordinates):
    thread_list = []
    num_of_rep = NUM_OF_REPETITIONS // NUM_OF_THREADS

    writeTimelaps('--- Preprocess {} ---'.format(datetime.now() - START_TIME))

    for thread_num in range(0, NUM_OF_THREADS):
        thread = Thread(target=worker, args=(thread_num, num_of_rep, vectorData,
                                             mapLayer, coordinates))
        thread_list.append(thread)

    for thread in thread_list:
        thread.start()
...
```

Před spuštěním vláken je počet pozorovacích míst, ze kterých je prováděna analýza viditelnosti, rozdělen rovnoměrně mezi definovaný počet vláken. Poté je zapsán čas, který byl spotřebován na přípravu testovacích dat a následně jsou spuštěna jednotlivá vlákna, která provedou všechny kroky analýzy stanovené ve *workeru*.

```
def worker(thread_num, vectorData, mapLayer, coordinates):
    start_time = datetime.now()
    coordinates = getCoordinates(ascii, thread_num)
    viewshedAnalysis(coordinates, thread_num)
    writeTimelaps('--- Thread {}: Elapsed Time {} ---'.format(thread_num,
                                            datetime.now() - start_time))
```

Po spuštění vláken jsou načteny souřadnice jednotlivých pozorovacích míst z ASCII souboru v jednotlivých vláknech, a následně je z těchto pozorovacích míst provedena analýza viditelnosti. Paralelní výpočet viditelnosti se od sekvenčního výpočtu liší pouze v cyklu v tom, že je výpočet rozdělen do 4 vláken. Parametry analýzy viditelnosti byly opět ponechány defaultně nastaveny s výškou pozorovatele 2,5 m.

```
def viewshedAnalysis(vectorData, mapLayer, coordinates, i, thread_num):
    printSync(thread_num, 'Computes the viewshed of a point on
                           an elevation raster map.')
    viewshed_map = vectorData + '_viewshed_thread{}_{}'.format(str(thread_num),
                                                               str(i))
    input_map_layer = '{}@{}'.format(mapLayer, MAPSET)
    for coordinates in coordinatesList:
    start_time = datetime.now()
    printSync(thread_num, 'Viewshed <{}@{}>'.format(viewshed_map, MAPSET))
    try:
        gscript.run_command('r.viewshed', input=input_map_layer,
            output=viewshed_map, overwrite=True,
            coordinates=(coordinates[0], coordinates[1]), flags='b')
    except:
        pass
    writeTimelaps('{} | {} | {}'.format(coordinates[0], coordinates[1],
                                        datetime.now() - start_time))
    return viewshed_map
```

Jakmile je dokončena analýza viditelnosti je zapsán výsledný čas paralelního výpočtu do textového souboru a poté jsou výsledné rastry z jednotlivých vláken sjednoceny do jednoho výsledného rastru viditelnosti *totalviewshed* prostřednictvím funkce *r.mapcalc*. Sjednocení je provedeno sekvenčně stejně jako v úloze bez paralelizace.

```
total_viewshed = '{}_totalviewshed'.format(vectorData)
viewshed_maps = []
for thread_num in range(0, NUM_OF_THREADS):
    viewshed_maps.append('{}_totalviewshed_thread{}'.format(vectorData,
                                                            thread_num))
totalviewshed(viewshed_maps, -1, total_viewshed)
```

Po dokončení sjednocení rastrů do jedné výsledné vrstvy je zapsán celkový čas výpočtu do textového souboru. Všechny časy, které jsou během výpočtu zapisovány do textového souboru vypadají následovně:

```
--- Preprocess 0:01:17.223000 ---
...
--- Thread 2: Elapsed Time 6:49:14.379000 ---
--- Thread 3: Elapsed Time 6:49:29.127000 ---
--- Thread 0: Elapsed Time 6:49:32.385000 ---
--- Thread 1: Elapsed Time 6:50:07.261000 ---
--- Elapsed Time 6:52:36.209000 ---
```

Obr. 20 Výsledný textový soubor s jednotlivými časy výpočtu

Čas, který byl spotřebován pro sečtení rastrů byl vypočten jako rozdíl času přípravy testovacích dat a času výpočtu posledního vlákna od celkového času výpočtu. Dosažené výsledky jsou vizualizovány a porovnány v následující kapitole 4.4.4. Celý skript je přiložen v příloze č. 3.

Stochastické vyhodnocení viditelnosti

Cíl úlohy stochastického vyhodnocení viditelnosti s paralelizací je totožný jako v předchozí úloze, a to dosáhnout lepších výsledků v porovnání se získanými výsledky úlohy stochastického vyhodnocení viditelnosti bez paralelizace.

Sekvenční část výpočtu úlohy zahrnuje veškeré předzpracování vstupujících vrstev do výpočtu viditelnosti, který je vyhodnocován metodou Monte Carlo. Tato část je shodná s částí předešlého výpočtu úlohy bez paralelizace. V rámci sekvenčního skriptu je definován import vstupních vrstev – vybraného místa pozorování (železniční stanice) prostřednictvím funkce *v.in.ogr* a digitálního výškového modelu (terénu) prostřednictvím funkce *r.in.gdal*. Následně byl nastaven region na naimportovaný terén pomocí funkce *g.region*.

Před paralelním zpracováním výpočtu viditelnosti byla dále upravena velikost terénu stejně jako v předchozí úloze. Kolem místa pozorování byla vytvořena obalová vrstva (buffer) o velikosti 10 000 m pomocí funkce *v.buffer*, převedena do rastrového formátu funkcí *v.to.rast* a následně touto vrstvou ořezán terén prostřednictvím rastrového kalkulátoru *r.mapcalc*.

Vyhodnocení viditelnosti již bylo provedeno paralelně ve 4 vláknech. Zahájení paralelního výpočtu bylo nadefinováno obdobně jako u předchozí paralelní úlohy. Než dojde k samotnému spuštění jednotlivých vláken, je počet opakování analýzy rozdělen dle počtu vláken, tedy jedno vlákno provede vyhodnocení analýzy 100x. Následně je zapsán čas, který byl spotřebován na přípravu testovacích dat pro stochastické vyhodnocení viditelnosti do textového souboru, a poté jsou spuštěna jednotlivá vlákna, která provedou analýzu pomocí funkce *target*.

```
def startThreads(vectorData, mapLayer, coordinates):
    thread_list = []
    num_of_rep = NUM_OF_REPETITIONS // NUM_OF_THREADS
    writeTimelaps('--- Preprocess {} ---'.format(datetime.now() -
                                                  START_TIME))
    for thread_num in range(0, NUM_OF_THREADS):
        thread = Thread(target=worker, args=(thread_num, num_of_rep,
                                vectorData, mapLayer, coordinates))
        thread_list.append(thread)
    for thread in thread_list:
        thread.start()
    ...
```

Stochastické vyhodnocení viditelnosti je provedeno stejně jako v předchozí úloze bez paralelizace. V prvním kroku je vytvořena prázdná rastrová vrstva pro ukládání jednotlivých variant výsledků viditelnosti (před cyklem). Poté je generován rastr náhodných chyb s rozptylem hodnot +-2,31 m dle EU-DEM viz kap. 4.4.1 (v cyklu). Poté je terén sečten s rastrem náhodných chyb z důvodu opravy terénu vlivem možného výskytu náhodných chyb (v cyklu) a následně je nad opraveným terénem provedena analýza viditelnosti.

```
def worker(thread_num, num_of_rep, vectorData, mapLayer, coordinates):
    start_time = datetime.now()
    terrainModification = 'terrain_modification_thread{}'.format(str(thread_num))

    varOfViewshed = produceRasterMapOfRandDeviates
            ('freqOfTerrain_thread{}'.format(str(thread_num)), 0, 0, thread_num)
    varOfViewshed = ''

    for i in range(0, num_of_rep):
        mistakes = produceRasterMapOfRandDeviates('mistakes_thread{}'.format
                                        (str(thread_num)), -2.3, 2.3, thread_num)

        expression = '{} = {}@{} + {}@{}'.format(terrainModification, mapLayer,
                                                  MAPSET, mistakes, MAPSET)
        rasterMapCalcNew(expression, thread_num)

        viewshed_map = viewshedAnalysis(vectorData, terrainModification,
                                        coordinates, i, thread_num)

        varOfViewshed = 'varOfViewshed_thread{}_{}'.format
                ;                   (str(thread_num), str(i))
        expression = '{} = {}@{} + {}@{}'.format(tmpVarOfViewshed,
                                varOfViewshed, MAPSET, viewshed_map, MAPSET)
        rasterMapCalcNew(expression, thread_num)
        varOfViewshed = tmpVarOfViewshed

    writeTimelaps('--- Thread {}: Elapsed Time {} ---'.format(thread_num,
                                                  datetime.now() - start_time))
```

Samotný výpočet viditelnosti je proveden stejně jako v první úloze prostřednictvím funkce *r.viewshed* s defaultně nastavenými parametry mimo výšky pozorovatele, která byla nastavena na 2,5 m s tím rozdílem, že je výpočet současně prováděn ve 4 vláknech (thread_num).

```
def viewshedAnalysis(vectorData, mapLayer, coordinates, i, thread_num):
     printSync(thread_num, 'Computes the viewshed of a point
                            on an elevation raster map.')
     viewshed_map = vectorData + '_viewshed_thread{}_{}'.format
                                            (str(thread_num), str(i))
     input_map_layer = '{}@{}'.format(mapLayer, MAPSET)
     start_time = datetime.now()
     printSync(thread_num, 'Viewshed <{}@{}>'.format(viewshed_map, MAPSET))
     try:
         gscript.run_command('r.viewshed', input=input_map_layer,
             output=viewshed_map, overwrite=True, coordinates=(coordinates[0],
                             coordinates[1]), observer_elevation=2.5, flags='b')
     except:
         pass
```

```
    writeTimelaps('{} | {} | {}'.format(coordinates[0], coordinates[1],
                                        datetime.now() - start_time))
    return viewshed_map
```

Jakmile všechna vlákna dokončí svou práci, zapíše se výsledný čas paralelního výpočtu každého vlákna do textového souboru a všechny výsledné rastry jsou poté sekvenčně sečteny do jednoho výsledného rastru viditelnosti prostřednictvím funkce *r.mapcalc*.

```
def rasterMapCalcNew(expression, thread_num):
    printSync(thread_num, 'Raster map calculator <{}>'.format(expression))
    gscript.run_command('r.mapcalc', expression=expression, overwrite=True,
                        quiet=True)
    return
...
freq = 'varOfViewshed_thread{}_{}'
expression = '{} = {}@{} + {}@{} + {}@{} + {}@{}'.format("varOfViewshed_final",
    freq.format(0, num_of_rep-1), MAPSET, freq.format(1, num_of_rep-1), MAPSET,
    freq.format(2, num_of_rep-1), MAPSET, freq.format(3, num_of_rep-1), MAPSET)
rasterMapCalcNew(expression, -1)
```

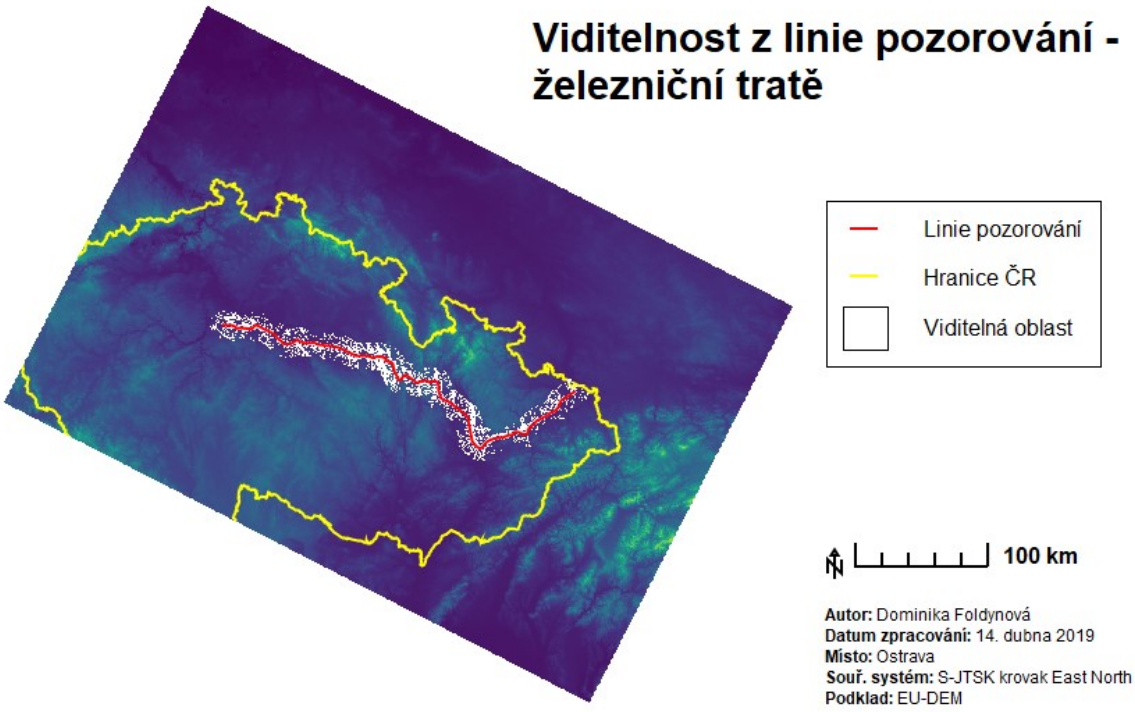
Po dokončení sjednocení rastrů do jedné výsledné vrstvy je zapsán celkový čas výpočtu do textového souboru. Čas, který byl spotřebován pro sečtení rastrů byl vypočten jako rozdíl času přípravy testovacích dat a času výpočtu posledního vlákna od celkového času výpočtu.

Celý skript zpracované paralelní i sekvenční části úlohy stochastického vyhodnocení viditelnosti je k dispozici viz příloha č. 4. Získané výsledky stochastického vyhodnocení viditelnosti na jednotlivých počítačích jsou vizualizovány a porovnány níže.

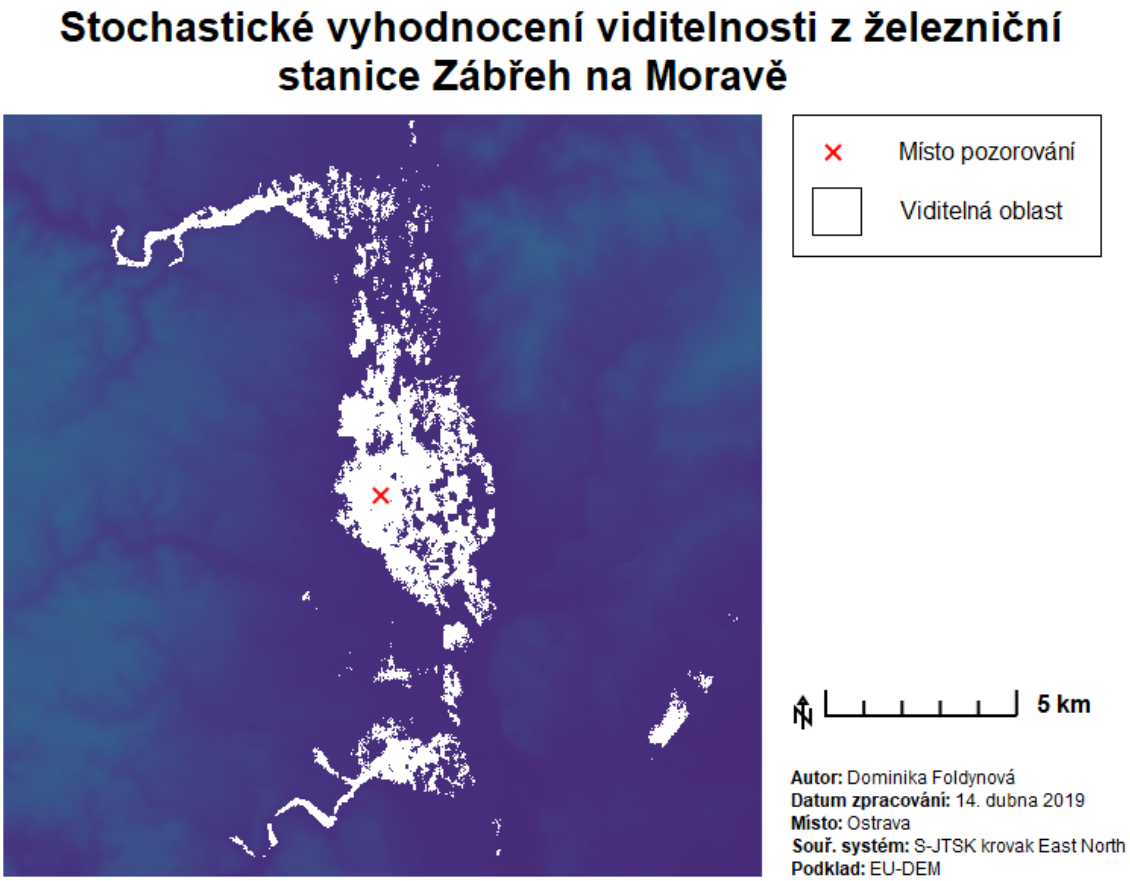
4.4.4 Srovnání výsledků

Výsledky obou úloh byly na závěr reklasifikovány pomocí funkce *r.reclass* a následně srovnány, zda jsou dosažené výsledky stejné, jak pro sekvenční výpočet, tak pro paralelní výpočet. Oba paralelní výpočty dosáhly stejných výsledků jako výpočty sekvenční. Výsledky obou úloh jsou vizualizovány na následujících obrázcích 22 a 23.

Pro snazší srovnání výsledků byly výsledky reklasifikovány do kategorií: neviditelná místa a místa viditelná alespoň z jednoho bodu linie (resp. alespoň v jednom z opakování Monte Carlo simulace).



Obr. 21 Výsledek pro obě řešení úlohy viditelnosti z linie



Obr. 22 Výsledek obou řešení stochastického vyhodnocení viditelnosti

Nicméně u obou úloh byla především sledována doba běhu výpočtu. Výsledky, kterých bylo dosaženo při výpočtu viditelnosti z linie bez paralelizace a s paralelizací na jednotlivých zmíněných počítačích (viz kap. 4.3) jsou vizualizovány v následujících tabulkách č. 2-3.

Tab. 2 Doba běhu výpočtu bez paralelizace

Úloha č. 1	
Počítač	**Doba běhu (T_S)**
PCJ424K	17:28:14
PCJ423V	21:33:14
PCJ339S	1 den, 2:10:24

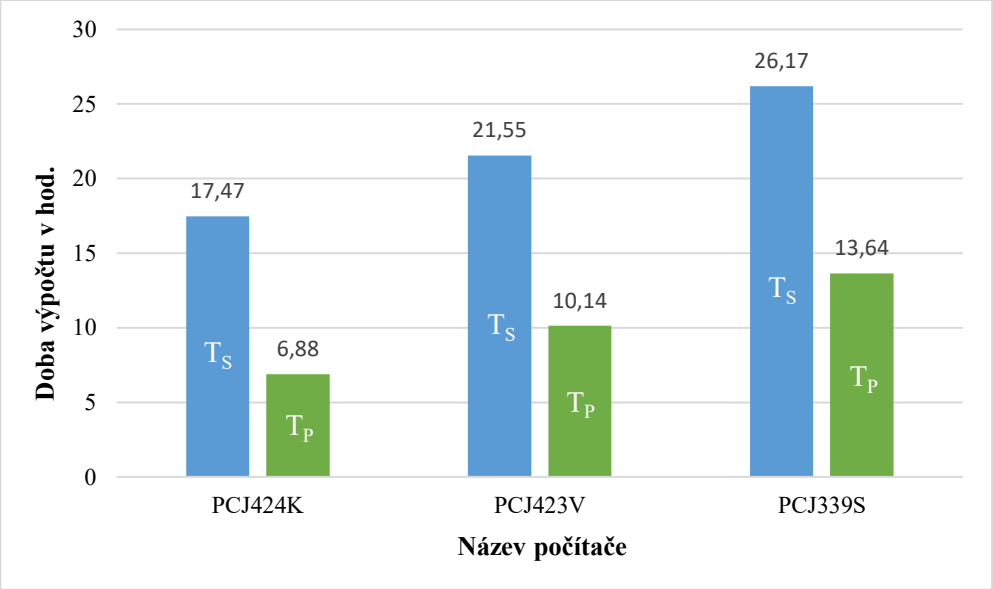
Nejrychleji byl proveden výpočet viditelnosti z linie bez paralelizace na počítači v učebně č. 424 s časem 17 hodin a 28 minut, který disponoval nejlepšími parametry (především pamětí).

Tab. 3 Doba běhu výpočtu s paralelizací

Úloha č. 1							
Počítač	**Příprava**	**Výpočet**				**Vyhodnocení**	**Doba běhu (T_P)**
		Vlákno č. 1	**Vlákno č. 2**	**Vlákno č. 3**	**Vlákno č.4**		
PCJ424K	0:01:17	6:49:32	6:50:07	6:49:14	6:49:29	0:01:12	6:52:36
PCJ423V	0:02:35	10:04:20	10:04:14	10:03:46	10:04:33	0:02:10	10:08:31
PCJ339S	0:03:25	13:32:05	13:32:11	13:32:09	13:31:52	0:02:31	13:38:07

Nejrychlejšího výpočtu viditelnosti z linie s paralelizací dosáhl opět stejný počítač „PCJ424K“ s časem výpočtu 6 hodin a 52 minut.

Výsledné časy všech provedených výpočtů byly vyneseny do grafu č. 1 a porovnány mezi sebou. Ve sloupcích modré barvy jsou vyneseny výsledné sekvenční časy a ve sloupcích zelené barvy jsou vyneseny výsledné paralelní časy. Všechny paralelní výpočty jsou ve výsledku rychlejší než výpočty, které byly provedeny sekvenčně.



Graf. 1 Doba běhu řešené úlohy č. 1

Výsledky, kterých bylo dosaženo při výpočtech úlohy stochastické vyhodnocení viditelnosti bez paralelizace a s paralelizací jsou zobrazeny níže v tab. 4 a 5.

Tab. 4 Doba běhu výpočtu bez paralelizace

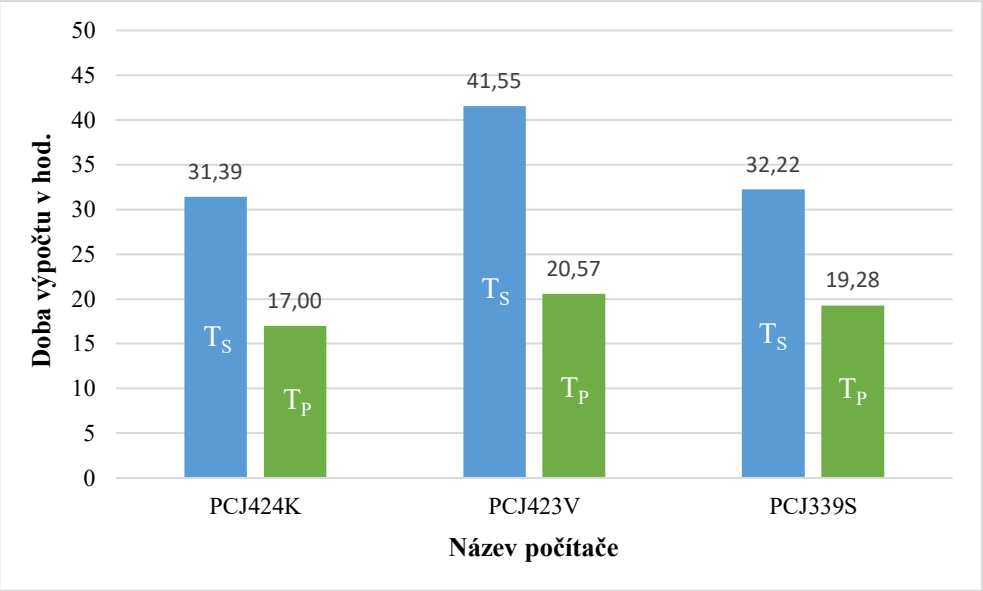
Úloha č. 2	
Počítač	**Doba běhu (T_S)**
PCJ424K	1 den, 7:23:23
PCJ423V	1 den, 17:32:55
PCJ339S	1 den, 8:13:18

Nejrychlejšího výpočetního času dosáhl počítač v učebně č. 424 stejně jako v předchozí úloze s výsledným časem 1 den 7 hodin a 23 minut při zpracování výpočtu bez paralelizace a stejně tak při výpočtu s paralelizací s časem necelých 17 hodin.

Tab. 5 Doba běhu výpočtu s paralelizací

Úloha č. 2							
Počítač	**Příprava**	**Výpočet**				**Vyhodnocení**	**Doba běhu**
		Vlákno č. 1	**Vlákno č. 2**	**Vlákno č. 3**	**Vlákno č.4**		**(T_P)**
PCJ424K	0:01:09	16:56:26	16:57:09	16:56:13	16:56:45	0:01:24	16:59:42
PCJ423V	0:02:26	20:29:56	20:29:54	20:29:54	20:29:57	0:02:03	20:34:26
PCJ339S	0:03:23	19:10:42	19:10:31	19:10:46	19:10:36	0:02:39	19:16:48

Výsledné časy všech provedených výpočtů úlohy č. 2 byly vyneseny do grafu č. 2 a porovnány mezi sebou. Ve sloupcích modré barvy jsou taktéž vyneseny výsledné sekvenční časy a ve sloupcích zelené barvy jsou vyneseny výsledné paralelní časy. Všechny paralelní výpočty jsou ve výsledku rychlejší než výpočty, které byly provedeny sekvenčně stejně jako u předchozí úlohy.



Graf. 2 Doba běhu řešené úlohy č. 2

Výsledné časy paralelních výpočtů lze hodnotit na základě několika parametrů, které byly zmíněny a popsány v kapitole 2.1. Na základě Amdahlova zákona (viz kap. 2) lze vypočítat dosažitelné zrychlení (S_1) v závislosti na počtu vláken, do kterých je paralelní výpočet rozdělen.

Tab. 6 Dosažitelné zrychlení

Počítač	**Úloha č. 1**	**Úloha č. 2**
	Zrychlení (S_1)	**Zrychlení (S_1)**
PCJ424K	3,98	3,97
PCJ423V	3,98	3,96
PCJ339S	3,98	3,94

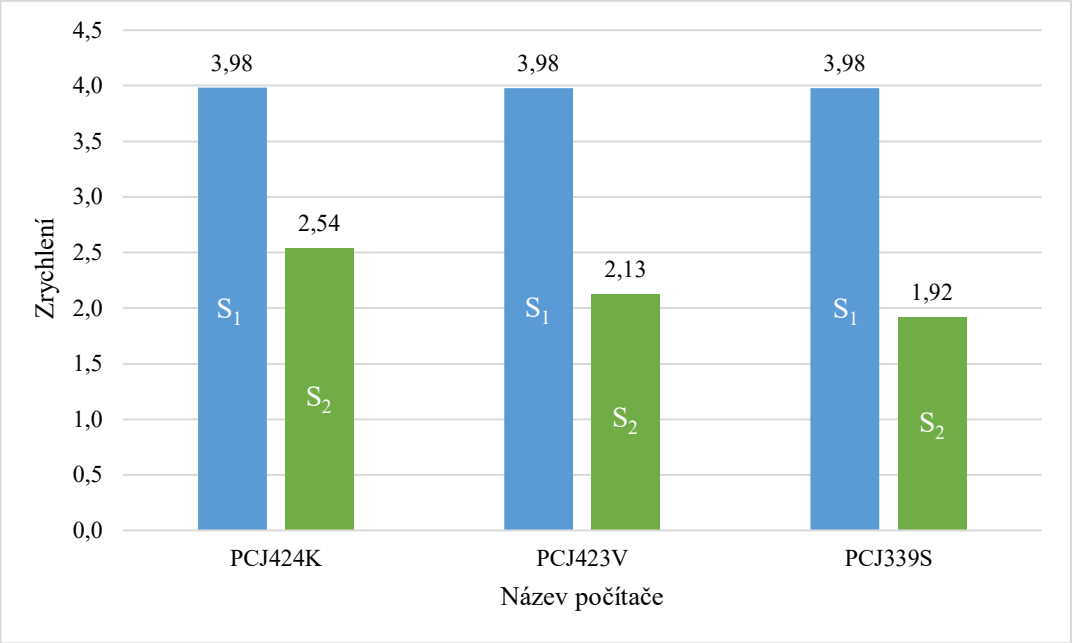
Dosažitelné zrychlení bylo následně porovnáno s dosaženým zrychlením (S_2), které udává kolikrát je výsledný paralelní algoritmus (T) rychlejší než nejlepší známý sekvenční algoritmus (SU).

$$S_2(n,p) = SU(n) \div T(n,p)$$

Tab. 7 Dosažené zrychlení

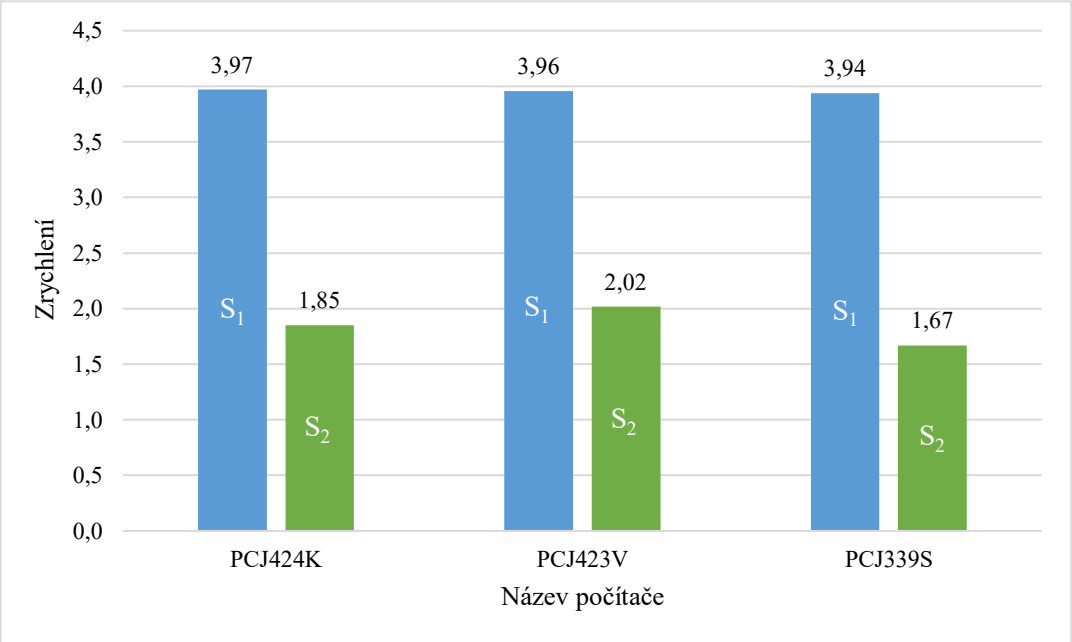
Počítač	**Úloha č. 1**	**Úloha č. 2**
	Zrychlení (S_2)	**Zrychlení (S_2)**
PCJ424K	2,54	1,85
PCJ423V	2,13	2,02
PCJ339S	1,92	1,67

Největšího zrychlení dosáhl počítač „PCJ424K“ při výpočtu úlohy č.1. Následující graf porovnává dosažitelné zrychlení s dosaženým zrychlením na základě výsledků paralelní úlohy č. 1.



Graf. 3 Porovnání dosažitelného zrychlení s dosaženým zrychlením pro úlohu č. 1

A graf č. 4 porovnává dosažitelné zrychlení s dosaženým zrychlením na základě výsledků paralelní úlohy č. 2.



Graf. 4 Porovnání dosažitelného zrychlení s dosaženým zrychlením pro úlohu č. 2

Paralelní výpočty byly dále hodnoceny na základě následujících parametrů:

- Paralelní cena – je součinem paralelního času a počtu procesorů.

Tab. 8 Výsledky parametru paralelní cena

Počítač	**Úloha č. 1**	**Úloha č. 2**
	Cena *(hod)*	**Cena *(hod)***
PCJ424K	27,52	68,00
PCJ423V	40,56	82,28
PCJ339S	54,56	77,12

- Paralelní efektivnost – je poměrem sekvenční a paralelní složitosti (ceny). Hodnoty leží intervalu <0, 1>, což splňují všechny výsledky viz tab. 9.

Tab. 9 Výsledky parametru paralelní efektivnost

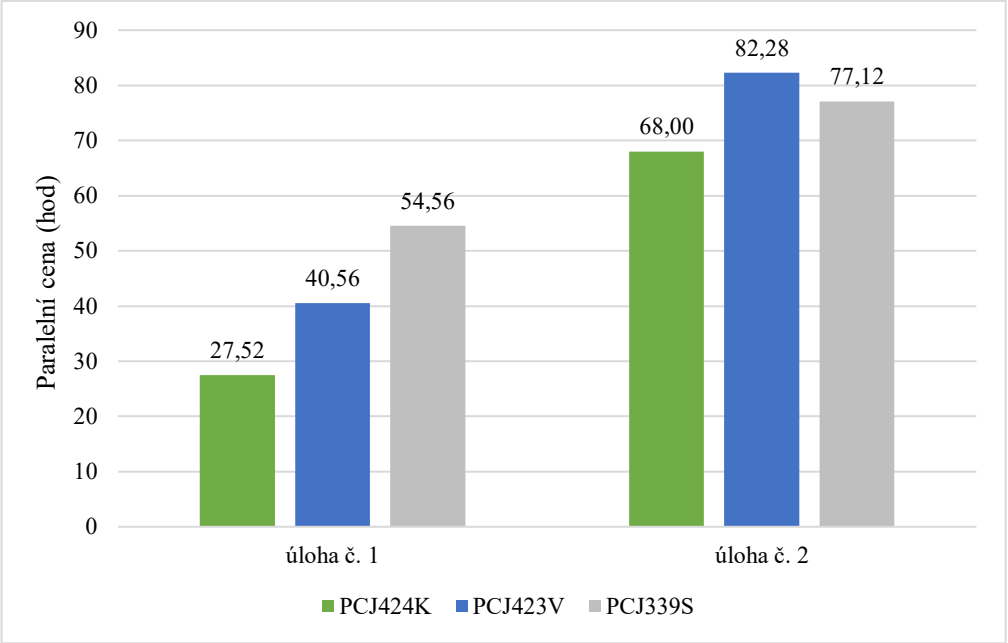
Počítač	**Úloha č. 1**	**Úloha č. 2**
	Efektivnost	**Efektivnost**
PCJ424K	0,64	0,46
PCJ423V	0,53	0,51
PCJ339S	0,48	0,42

- Paralelní režie – je rozdíl paralelní ceny a času nejrychlejšího známého sekvenčního algoritmu.

Tab. 10 Výsledky parametru paralelní režie

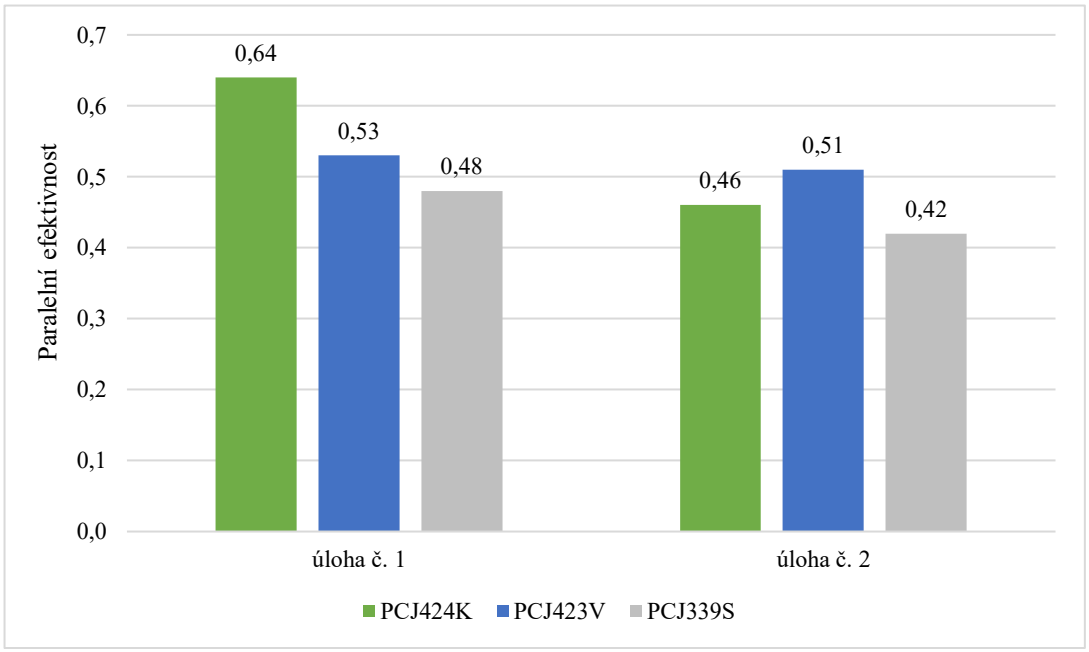
Počítač	**Úloha č. 1**	**Úloha č. 2**
	Režie *(hod)*	**Režie *(hod)***
PCJ424K	10,50	36,61
PCJ423V	19,01	40,73
PCJ339S	28,39	44,90

Výsledné hodnoty všech vyhodnocených parametrů byly vyneseny do grafu a porovnány mezi sebou (viz grafy 5-7).



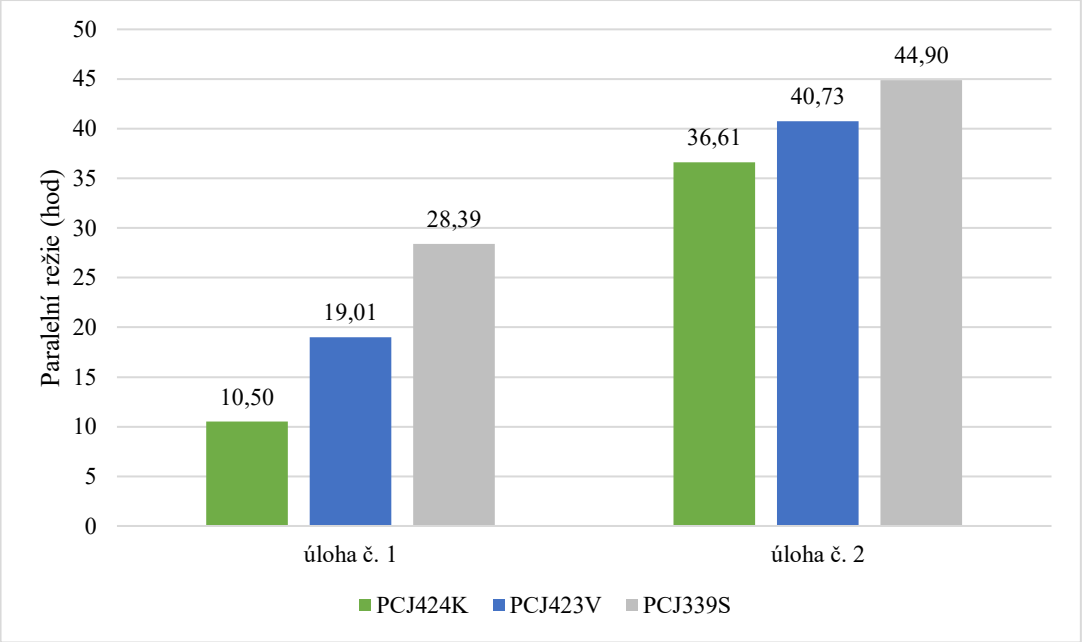
Graf. 5 Srovnání výsledků paralelní ceny

Nejnižší ceny dosáhl počítač „PCJ424K“ při výpočtu úlohy č. 1 s paralelní cenou 27,52 hod. Naopak nejvyšší paralelní ceny dosáhl počítač „PCJ423V“ při výpočtu úlohy č. 2 s celkovou paralelní cenou 82,28 hod.



Graf. 6 Srovnání výsledků paralelní efektivnosti

Největší paralelní efektivity dosáhl počítač „PCJ424K“ při výpočtu úlohy č. 1 s celkovou paralelní efektivitou 0,64.



Graf. 7 Srovnání výsledků paralelní režie

Nejnižší paralelní režie dosáhl počítač „PCJ424K“ při výpočtu úlohy č. 1 s celkovou paralelní režií 10,50 hod.

5 ZHODNOCENÍ

Cílem diplomové práce bylo nalézt vhodnou metodu paralelizace výpočtu analýzy viditelnosti s ohledem na všechna omezení související s analýzou (viz kap. 4.2), a následně vybranou metodu otestovat na dvou vybraných úlohách – viditelnost z linie a stochastické vyhodnocení viditelnosti metodou Monte Carlo. Pro obě řešené úlohy, které provádí analýzu viditelnosti, byl vyhodnocen jako nejvhodnější datový paralelismus. A to z toho důvodu, že dělení dat je spíše na straně datové sady míst pozorování (u první úlohy – viditelnosti z linie). Dělení podkladového terénu je složitější, protože kvadranty, do kterých je terén rozdělován, se překrývají. Při výpočtu obou úloh byl zpracováván rastr o velikosti 503 098 065 px. V obou řešených úlohách se výsledek sekvenčního vyhodnocení úlohy shoduje s výsledkem paralelního vyhodnocení téže úlohy.

V úvodu druhé kapitoly bylo zmíněno, že ne vždy platí přímá úměra mezi počtem výpočetních jednotek a rychlostí zpracování, i když je logické si myslet, že výpočet, který je rozložen do 4 vláken bude 4x rychlejší. Výsledky paralelizace totiž ovlivňuje řada faktorů. V tomto případě bylo dosaženo nejlepšího paralelního času, který byl 2,5x rychlejší než čas sekvenčního výpočtu. Paralelní zrychlení není v tomto případě přímo úměrné z důvodu brzdění výpočtu kvůli I/O operacím – zápisu a čtení z disku.

Obecně lze říci, že problematika náročnosti takto rozsáhlých výpočtů je složitá. Pro dosažení nejlepších výsledků záleží na vhodně zvolené metodě paralelizace výpočtu, zda bude výpočet proveden ve více vláknech (multithreding) nebo na více procesorech (multitasking), a do kolika vláken příp. procesorů bude úloha, kterou lze paralelizovat rozložena.

Složitost výpočtu obou úloh by rostla v případě, kdyby do analýzy viditelnosti vstupoval rozsáhlejší terén, a proto z hlediska časové náročnosti a vytíženosti počítačů, které byly využity v rámci diplomové práce, byly provedeny výpočty pouze v jedné variantě s velikostí terénu do 10 000 m linie pozorování, či místa pozorování.

6 ZÁVĚR

Diplomová práce se zabývá problematikou náročnosti výpočtů viditelnosti a jejich paralelizací v jazyce python metodou datového paralelismu ve více vláknech v programu GRASS GIS. Na začátku se práce zabývá paralelními výpočty, možnostmi paralelizace, možností rozložení paralelního výpočtu do více vláken nebo procesorů. Mezi stěžejní části této sekce patřil popis možných přístupů a způsobů, jak rozdělit práci mezi více vláken. Ve druhé části teoretické části je popsána problematika náročnosti výpočtů viditelnosti, jaké metody analýzy viditelnosti existují, a kdo a jak se touto problematikou zabýval.

V úvodu praktické části jsou popsány vybrané úlohy, jejichž problematika byla řešena v rámci již obhájené závěrečné práce a na cvičení v rámci předmětu Modelování a simulace v geovědách. Pro každou úlohu byly následně popsány možnosti, jak je lze paralelizovat a jak je lze rozdělit do více vláken. Vybrané metody byly otestovány na testovacích datech, které byly upraveny a použity při výpočtech.

Dále se praktická část věnuje popisu obou řešení testovaných úloh, kde při testování byl kladen důraz na zrychlení oproti sekvenčnímu provedení úlohy. Jednotlivé výsledky všech výpočtů byly vizualizovány a popsány nejen slovně ale i graficky. Při srovnání výsledků bylo zjištěno, že nejlepších výsledků dosáhl počítač s nejlepší konfigurací.

Při případném rozšiřování této úlohy by bylo vhodné otestovat algoritmy na větším rozsahu terénu s rozložením paralelního výpočtu do více vláken pro demonstraci možného zrychlení algoritmů, či případného zpomalení, ke kterému může dojít z důvodu objemnější komunikace mezi vlákny.

SEZNAM LITERATURY A ZDROJŮ

BLIZŇÁK, Michal. 2014. *Úvod do paralelismu, metody paralelizace algoritmů* [online]. Zlín [cit. 2019-02-25]. Dostupné z: https://vyuka.fai.utb.cz/pluginfile.php/37792/mod_resource/content/0/Lectures/2014/Lecture4.pdf

DOHNALOVÁ, Martina. 2012. *Citlivostní analýza vyhodnocení viditelnosti z linie v ArcGIS.* Ostrava. Diplomová práce. VŠB – Technická univerzita Ostrava.

DUFOURMONT, et al. 2014. In: *EU-DEM Statistical Validation Report* [online]. Německo: DHI Business Management System [cit. 2019-04-13].

DUMEK, V. 2003. *Operační systémy* – Pomocný studijní text pro předmět vyučovaný ve čtvrtém ročníku magisterského studia. VUT Brno, Brno.

DVORSKÝ, Jiří. 2012. *Analýza a zpracování rozsáhlých grafů* [online]. Ostrava [cit. 2019-02-25]. Dostupné z: https://innet.vsb.cz/personCards/solverProjects.jsp?projectDetailId=37245&login=dvo26&lang=cs

ESRI. 2016. *Using Viewshed and Observer Points for visibility analysis* [online]. Environmental Systems Research Institute [cit. 2019-03-03]. Dostupné z: http://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/using-viewshed-and-observer-points-for-visibility.htm#

FAIGL, Jan. 2016. *Vícevláknové aplikace* [online]. Praha [cit. 2019-02-25]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/a0b36pr2/lectures/lecture05-slides.pdf

FIALA, P., P. KARHAN a J. PTÁČEK. 2014. *Neuronové sítě a možnosti jejich využití* [online]. Olomouc [cit. 2019-02-25]. Dostupné z: http://www.csfm.cz/userfiles/file/Udalosti_2014/KRF2014/fiala.pdf

GOVE, Darryl. 2011. *Programování aplikací pro vícejádrové procesory*. 1. Brno: Computer Press. ISBN 978-80-251-3487-0.

GRASS GIS. 2015. Portál FreeGIS [online]. GNU Operating System, 1990 [cit. 2019-04-11]. Dostupné z: http://freegis.fsv.cvut.cz/gwiki/GRASS_GIS

JAKL, Ondřej. 2011. *Úvod do programování paralelních systémů*. 1. Liberec: Technická univerzita. ISBN 978-80-7372-715-4.

KAJGR, Václav. 2016. *Modelování vlivu vegetace na viditelnost území*. Ostrava. Diplomová práce. VŠB – Technická univerzita Ostrava.

KLEMENT, Milan. *Počítačové komponenty*. Olomouc: Univerzita Palackého v Olomouci, 2015. ISBN 978-80-244-4567-0.

KOKTAN, Jiří. 2017. *Paralelizace výpočetních postupů ve stavební mechanice* [online]. Ostrava [cit. 2019-02-25]. Dostupné z: https://www.fast.vsb.cz/228/cs/resene-projekty/?projectDetailId=41069

KOKTAN, Jiří. 2018. *Paralelní algoritmy pro analýzu desek na podloží* [online]. Ostrava [cit. 2019-02-25]. Dostupné z: https://www.fast.vsb.cz/228/cs/resene-projekty/?project DetailId=43489

LARYŠ, Kryštof. 2011. *Paralelní programování v .NET Frameworku*. Ostrava. Diplomová práce. VŠB – Technická univerzita Ostrava.

LELEK, Vojtěch. 2016. *Pokročilé vyhodnocení viditelnosti z linie*. Ostrava. Bakalářská práce. VŠB – Technická univerzita Ostrava.

MATĚJOVIC, Přemysl. 1997. *Paralelní architektury a programy*. 1. Plzeň: ZČU. ISBN 80-7082-322-4.

OLEJ, Vladimír a Petr HÁJEK. 2009. *Úvod do umělé inteligence: klasická umělá inteligence: distanční opora*. Pardubice: Univerzita Pardubice. ISBN 978-80-7395-241-9.

POLÁŠEK, Jaromír. 2016. *Využití paralelizovaných matematických operací v oblasti zpracování dat*. Brno. Bakalářská práce. Vysoké učení technické v Brně.

POPELKA, Stanislav. 2012. *Analýzy viditelnosti* [online]. Olomouc [cit. 2019-01-25]. Dostupné z: http://kartometrie.upol.cz/materialy/06_prednaska.pdf

PRIŠŤ, Lukáš. 2013. *Využití paralelizovaných výpočtu v prostředí MATLAB ve zpracování obrazu*. Brno. Bakalářská práce. Vysoké učení technické v Brně.

RÁŠOVÁ, Alexandra. 2015. *Substanciálna analýza viditel'nosti v prostredí GIS*. Bratislava. Písemná část dizertační práce. VŠB – Technická univerzita Ostrava.

RŮŽIČKOVÁ, Kateřina. 2017. *Vliv nejistoty terénu na výsledky analýzy – stanovení odtokové sítě*. Ostrava.

ŽÁK, Michal. 2016. *Jak se měří a kdy je nejlepší dohlednost* [online]. Dostupné z: https://www.in-pocasi.cz/clanky/teorie/dohlednost-8.12.2016/. [cit. 2019-04-14].

SEZNAM OBRÁZKŮ

SEZNAM GRAFŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Tištěné přílohy:

Příloha č. 1: Skript řešení úlohy č. 1 – viditelnost z linie bez paralelizace

Příloha č. 2: Skript řešení úlohy č. 1 – viditelnosti z linie s paralelizací

Příloha č. 3: Skript řešení úlohy č. 2 – stochastické vyhodnocení viditelnosti bez paralelizace

Příloha č. 4: Skript řešení úlohy č. 2 – stochastické vyhodnocení viditelnosti s paralelizací

Přílohy na CD:

Textová část diplomové práce

Upravená testovací data

Tištěné přílohy č. 1–4 v elektronické podobě